uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería en Tecnologías Industriales

Curso académico 2017-2018

Trabajo Fin de Grado

"Localización de vehículos utilizando sensores 3D y mapas"

José Suárez Muñoz

Tutor:

Arturo de la Escalera Hueso

Lugar y fecha de presentación prevista- *04/07/2018*

RESUMEN

La presente memoria recoge el desarrollo de un algoritmo de localización de vehículos autónomos en entornos urbanos, basándose en la información obtenida de sensores 3D y de mapas libres.

En esta memoria se incluye un análisis del marco regulador y del impacto socio-económico del proyecto.

Dentro del desarrollo del proyecto el fundamento del algoritmo es la detección de edificios tanto por los sensores 3D como por los mapas libres en una ubicación dada, utilizando los planos correspondientes a estos edificios para determinar la ubicación del vehículo en cualquier punto.

Tanto para los mapas libres (OpenStreetMap) como para la información de los sensores 3D (nubes de puntos) se realizan, por separado, un conjunto de pruebas iniciales para analizar sus resultados en casos sencillos, aumentando la dificultad de estos conforme se avanza en el desarrollo.

Después de completar las pruebas iniciales se unen ambos resultados y se realizan las pruebas finales en las que se obtiene la ubicación.

Finalmente, se mapea una zona del mapa con el recorrido del vehículo a lo largo de una trayectoria y se representan los errores cometidos.

Palabras clave: *sensores 3D, nubes de puntos, vehículos autónomos*

AGRADECIMIENTOS

Me gustaría comenzar expresando mi agradecimiento a todos y cada uno de los que me han apoyado en la elaboración de este proyecto, en particular a mi padre y a mi madre, que llevan apoyando durante los cuatro años que he cursado el Grado, día a día tanto en los buenos como en los malos momentos. Sin duda, sin ellos no habría llegado hasta donde estoy.

También me gustaría agradecer al resto de mi familia, porque siempre han estado cuando los he necesitado.

Agradecer también a mi actual pareja, que además de conseguir sacarme una sonrisa cada día tiene el mérito de haberme soportado en los momentos más difíciles.

A mi tutor, que me ha orientado desde el primer momento, mostrándose siempre dispuesto a ayudar y convirtiéndose en el principal motivo por el cual me decanté por este proyecto.

CONTENIDO

INDICE DE FIGURAS

1. INTRODUCCIÓN

En la sociedad actual el automóvil es el principal medio de transporte utilizado en todo el mundo y, a pesar del avance de la tecnología, el uso de estos vehículos supone un riesgo importante para la salud.

Si se comparan los vehículos de hace unas décadas con los actuales, se puede comprobar que estos últimos emiten una cantidad menor de gases contaminantes y a la vez son más seguros en caso de accidentes. Sin embargo, debido al gran número de automóviles existentes, la emisión total de estos gases es uno de los principales motivos del cambio climático. Además, otro factor muy importante que se debe tener en cuenta es el número de accidentes registrados cada año, el cual sigue siendo muy elevado.

En los últimos años se han llevado a cabo numerosas investigaciones con el objetivo de reducir el riesgo en el uso de estos vehículos.

Por un lado, el desarrollo de vehículos híbridos y eléctricos supone una reducción muy importante en la contaminación, al no depender exclusivamente de combustibles fósiles.

Por otro lado, con el fin de reducir el número de accidentes y mejorar la circulación se han desarrollado los vehículos autónomos.

Por autónomo se entiende un vehículo el cual no necesita de un conductor para poder circular libremente. Esto es posible debido a un gran número de componentes integrados en él, tanto hardware como software, que almacenan y procesan información del exterior.

Sin embargo, en los últimos años se han desarrollado distintos proyectos que impulsan el uso de estos sistemas, como por ejemplo el proyecto "*The KITTI Vision Benchmark Suite"* llevado a cabo por el Instituto Karlsruhe de Tecnología (KIT) y el Instituto Tecnológico de Toyota en Chicago.

El problema de adaptar los sistemas de posicionamiento de los vehículos no autónomos a los que sí lo son surge en zonas con una gran presencia de obstáculos, como por ejemplo árboles, túneles, edificios… [1].

Estos obstáculos suponen una pérdida en la señal recibida por el GPS procedente de los distintos satélites. En un vehículo autónomo, donde la conducción se rige por la información recibida por el exterior, la pérdida de señal GPS durante unos segundos supone un gran peligro, tanto para los pasajeros como para los que le rodean.

1.1 Objetivo y Motivación

El objetivo principal de este proyecto es la implementación de un sistema, basado en sensores 3D y mapas, que sea capaz de ubicar un vehículo autónomo circulando por entornos urbanos. Para ello se emplean distintas herramientas, las cuales serán explicadas en apartados posteriores, como OpenStreetMap y Point Cloud Library.

Además de las herramientas empleadas, se van a utilizar los datos disponibles en el proyecto mencionado en el apartado anterior *KITTI Vision Benchmark Suite*, para obtener las imágenes y las nubes de puntos necesarias en el desarrollo del trabajo.

El fundamento en el que se basa este proyecto es en la presencia y detección de edificios en la carretera, tanto por parte de la nube de puntos como por OpenStreetMap, con el objetivo de unir ambos resultados y utilizar los planos de las paredes de estos edificios para localizar el vehículo en su recorrido por la ciudad.

Para ello, se han ido realizando numerosas pruebas, desde pruebas iniciales muy sencillas para comprobar el funcionamiento de las librerías y de las herramientas, hasta las pruebas finales donde se mapea el recorrido del vehículo.

Por tanto, primero se explicará cómo se han instalado estas librerías y herramientas y cómo se han obtenido los datos necesarios.

A continuación, se realizarán las pruebas iniciales por separado de cada librería y una vez comprobado su correcto funcionamiento se unirán en un mismo código para obtener los resultados finales.

La motivación para la realización de este Trabajo de Fin de Grado viene impulsada por el interés en los temas relacionados con conducción autónoma, sensores y programación.

Respecto a la conducción autónoma, no hay ninguna duda de que en unos años será el método implantado en la mayoría de los medios de transporte, y el hecho de investigar y desarrollar un proyecto centrado en este tema refuerza la motivación de realizarlo, sabiendo que los resultados obtenidos en él pueden ser útiles en un futuro.

En cuanto a la programación, aunque durante el Grado no se haya profundizado en ella, es una disciplina que siempre me ha llamado la atención y este proyecto presenta una gran oportunidad para obtener más conocimientos en el lenguaje utilizado, en este caso *C++*.

1.2 Marco Regulador

La idea de conducción autónoma es un concepto que nació unos pocos años atrás, y que poco a poco ha ido desarrollándose sin llegar a estar completamente terminada.

Debido a esto, no existe actualmente un marco regulador en la Unión Europea ni en España que establezca las normas a seguir en el desarrollo e implantación de la conducción autónoma.

En noviembre de 2015, la DGT (*Dirección General de Tráfico*) estableció un marco para la realización de pruebas con vehículos de conducción automatizada en vías abiertas a la circulación.

En ella se recogen de forma detallada los requisitos necesarios para poder solicitar estas pruebas, al igual que una clasificación de estos vehículos en función de su nivel de automatización.

Según este marco regulatorio, la definición de vehículo autónomo es: "todo aquel que dispone de capacidad motriz equipado con tecnología que permita su manejo o conducción sin precisar la forma activa de control o supervisión de un conductor, tanto si dicha tecnología autónoma estuviera activada o desactivada de forma temporal o permanente." [2].

En torno a la Unión Europea, en abril de 2016 se firmó el primer documento sobre este tema: la Declaración de Ámsterdam sobre cooperación en el campo de la conducción automatizada y conectada, cuyo objetivo era promover un marco normativo para el año 2019 [3].

Por otro lado, el lenguaje de programación utilizado en este proyecto es C++11, la cual es una versión del estándar C++ aprobado por la ISO el 12 de agosto de 2011.

Respecto a las herramientas utilizadas, todas ellas son de libre uso con fines académicos y de investigación, siempre y cuando no se utilicen con ánimo de lucro.

En el apartado 3 de la memoria se describen estas herramientas y se explican las licencias de cada una de ellas.

1.3 Entorno socio-económico

En este apartado se pretende realizar un análisis del entorno socio-económico de la aplicación del resultado del proyecto.

Como se ha mencionado en la introducción de la memoria, los accidentes relacionados con vehículos siguen siendo muy elevados, a pesar de los avances en la tecnología y seguridad de los mismos.

El objetivo de la conducción autónoma es minimizar o eliminar en la medida de lo posible estos accidentes, que suponen un riesgo muy elevado para las personas, tanto conductores como peatones.

Un aspecto muy importante de la conducción autónoma es la localización, ya que el vehículo debe conocer su posición en cualquier instante de tiempo para poder circular de forma segura, tema en el cual se profundiza en este proyecto.

Por tanto, este trabajo al ser de carácter teórico podría utilizarse en aplicaciones de localización de vehículos autónomos.

Respecto al impacto social, el conseguir aumentar la seguridad de la conducción es un objetivo que sería muy bien recibido por parte de la sociedad, permitiendo la conducción a cualquier persona independientemente de sus condiciones físicas, como movilidad reducida.

Además, los vehículos autónomos pretenden utilizar energías renovables, como electricidad, sustituyendo a los combustibles fósiles, lo cual tendría un impacto medioambiental positivo disminuyendo la contaminación emitida a la atmósfera.

Sin embargo, el impacto ético podría no estar definido, ya que en caso de ocasionar un accidente habría distintas opiniones éticas acerca de si fue ocasionado por el conductor, por el peatón, por fallo de los sistemas del vehículo… siendo motivo de amplios debates.

1.3.1 Presupuesto de elaboración del Trabajo de Fin de Grado

En este apartado se procede a detallar el presupuesto de elaboración del proyecto, realizando una estimación del coste.

Respecto a los recursos empleados, en este proyecto se ha hecho uso de un ordenador portátil Acer Aspire E1-572G de 5 años de antigüedad, por lo que no se invirtió en un ordenador nuevo, y únicamente se tiene en cuenta el coste de amortización del ordenador.

El coste total del ordenador fue de 500€, por lo que se puede considerar un coste de amortización en el desarrollo del proyecto de 75€:

$$c_{A,O} = \frac{500€}{5\ años} \cdot \frac{3}{4} años = 75€ \tag{1.1}$$

En cuanto a las horas empleadas, los ECTS correspondientes a la realización del proyecto son 12. Cada crédito ECTS equivale a 25 horas de trabajo del estudiante, resultando en un número total de 300 horas.

Estimando un coste por hora de 8€ aproximadamente, el coste horario es:

$$c_H = 300\ horas \cdot 8 \frac{€}{hora} = 2400€ \tag{1.2}$$

Debido a que es un proyecto teórico, no se han utilizado más recursos por lo que el presupuesto final es:

$$Presupuesto = c_H + c_{A,O} = 2475€ \tag{1.3}$$

2. ESTADO DEL ARTE

El desarrollo de los vehículos autónomos ha experimentado una gran evolución en los últimos años, convirtiéndose en un campo de investigación muy amplio e importante en el cual hay numerosos programas activos llevados a cabo por empresas como *Renault*, *Ford*, *Bosch*…

2.1 Historia y evolución

La idea de concebir un vehículo que sea capaz de conducir de forma autónoma, es decir, sin la necesidad de un conductor, tuvo su origen a principios del siglo XX y no fue hasta 1925 cuando se desarrolló el primer vehículo autónomo.

La empresa Houdina Radio Control ideó un vehículo conducido por un sistema de radiocontrol, llamado *American Wonder (Fig. 2.1)*.

Este vehículo se guiaba por las órdenes que recibía en su antena, procedentes de otro vehículo situado a varios metros, haciendo uso de un transmisor. [4]

El fundador de esta empresa organizó un evento público el 8 de diciembre de 1926 para demostrar el funcionamiento del vehículo en las calles de Nueva York, recorriendo el trayecto que separa Broadway y la Quinta Avenida.

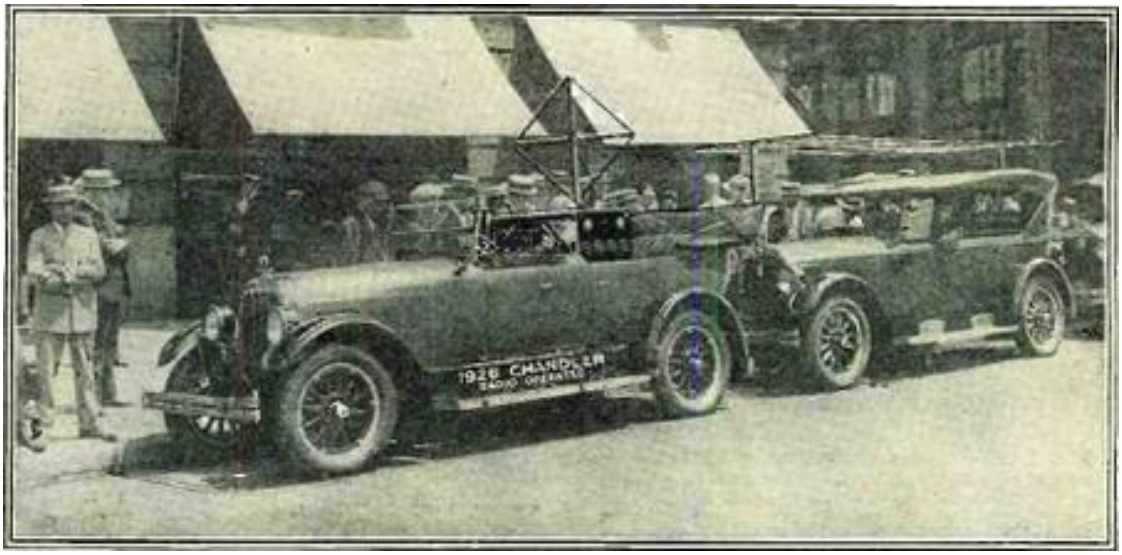


Fig. 2.1 Fotografía del vehículo americano American Wonder

Este hecho marcó el punto de inicio de los vehículos autónomos, basados en el radiocontrol, y el siguiente evento importante tuvo lugar en 1939, en la Exposición Universal de Nueva York.

En esta exposición, patrocinada por General Motors, el diseñador industrial Norman Bel Geddes presentó una maqueta de una ciudad del mañana, en la cual los automóviles eran eléctricos, y se movían impulsados por un campo electromagnético procedente de un circuito eléctrico insertado en el pavimento (*Fig. 2.2*) [5].

Fig. 2.2: Maqueta de la ciudad del mañana diseñada por Norman Bel Geddes en 1939

En 1980, el alemán Ernst Dickmanns junto con su equipo de la Universidad de Múnich diseñaron una furgoneta Mercedes Benz (*Fig. 2.3*) controlada por una cámara que procesaba las imágenes captadas del exterior para dirigir el vehículo. Como dato, esta furgoneta alcanzó la velocidad de 100 kilómetros por hora en calles sin tráfico.

Fig. 2.3: Furgoneta autónoma Mercedes Benz diseñada por Ernst Dickmanss en 1980

En ese mismo año, la agencia DARPA (*Defense Advanced Research Projects Agency)* desarrolló el primer vehículo que utilizaba técnicas de visión por computador y radares láser.

Más tarde, en 1994, Ernst Dickmanns modificó dos Mercedes Benz Clase S con el objetivo de circular en París a una velocidad de 130 kilómetros por hora. (*Fig. 2.4).*

Para ello, los vehículos disponían de visión computerizada que era procesada por un ordenador para poder reaccionar en tiempo real. Mediante cuatro cámaras se consiguió que estos coches reconocieran hasta doce vehículos. [6].

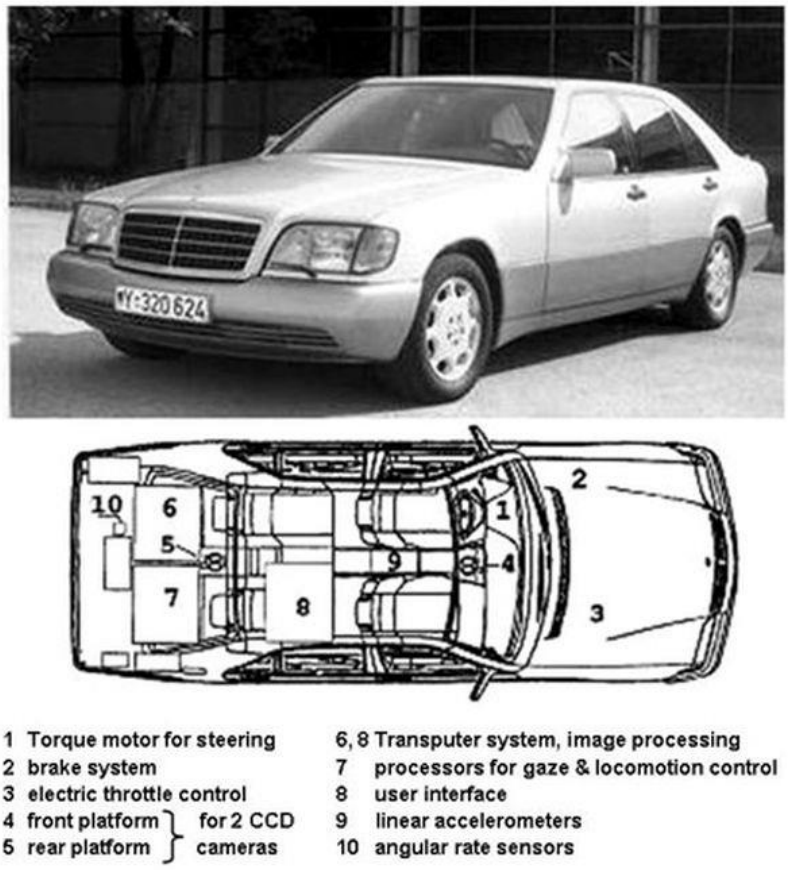


Fig. 2.4: Mercedes Benz Clase S modificado por Ernst Dickmanns en 1994

Ya en tiempos más modernos, Audi diseñó en 2014 un modelo RS7 autónomo que alcanzó los 240 kilómetros por hora en el circuito alemán de Hockenheim. Además, un piloto profesional dio una vuelta al circuito con el mismo coche siendo 5 segundos más lento que el no tripulado.

2.2 Estado actual de los vehículos autónomos

Actualmente hay numerosos proyectos activos dedicados al desarrollo de los vehículos autónomos, pero antes de enumerarlos es necesario distinguir entre los distintos niveles de conducción autónoma que existen.

La Sociedad de Ingenieros de Automoción (*SAE)* realizó una división distinguiendo seis niveles: [7]

- **Nivel 0***:* Cuando el vehículo no tiene ningún sistema automatizado que le permita tomar el control.

- **Nivel 1:** En este nivel se incluyen los vehículos con tecnologías como control de crucero o mantenimiento de carril.

- **Nivel 2:** El vehículo incluye sistemas de automatización de la conducción que permiten el control del movimiento longitudinal y lateral. Este sistema se desactiva cuando el conductor toma el control del coche.

- **Nivel 3:** Los vehículos pueden circular de forma autónoma en entornos controlados como autopistas. Estos vehículos incluyen detección y respuesta ante objetos. Sin embargo, sigue siendo necesaria la presencia de un conductor, el cual puede tomar el control del vehículo en cualquier momento.

- **Nivel 4:** El vehículo puede circular sin necesidad de la supervisión de un conductor en áreas con información suficiente, pudiendo encontrarse situaciones donde no sea capaz de circular autónomamente.

- **Nivel 5:** El vehículo es completamente autónomo, pudiendo circular libremente por cualquier carretera donde se permita la conducción autónoma.

Entre los proyectos más destacados se encuentra el prototipo desarrollado por Google, el cual utiliza un software de navegación probado en automóviles Lexus. Estos últimos

recorrieron un total de más de un millón de kilómetros de forma autónoma siendo supervisados por conductores.

Otra de las principales empresas que está desarrollando vehículos autónomos es General Motors con su vehículo Cruise AV de nivel 4 de conducción autónoma.

2.2.1 Estado actual de los vehículos autónomos en visión artificial

Uno de los temas más importantes relacionados con los vehículos autónomos es la visión artificial.

Estos vehículos deben de ser capaces de detectar todo lo que les rodea en el entorno para reaccionar ante los objetos, además de ser capaces de utilizar esta información para ubicarse en un mapa.

Entre los distintos proyectos activos dedicados a esta materia cabe destacar el desarrollado por el Instituto de Tecnología de Karlsruhe (*KIT)* en colaboración con el Instituto Tecnológico de Toyota, denominado "*The KITTI Vision Benchmarck Suite"*, disponible en [8].

Sus áreas de investigación son flujo estéreo y óptico, odometría visual y detección de objetos 3D.

Para ello, se ha equipado a una furgoneta Volkswagen Passat (*Fig. 2.5)* de dos cámaras de vídeo de alta resolución, un escáner láser Velodyne y un sistema de localización GPS.

Fig. 2.5: Componentes Volkswagen Passat utilizado en el proyecto KITTI Vision Benchmark Suite

Los datos se han recopilado conduciendo el vehículo por la ciudad de Karlsruhe, tanto en zonas urbanas como rurales, estando disponibles en [8].

2.3 Componentes de los vehículos autónomos

Los vehículos autónomos tienen implantados numerosos componentes, dividiéndose en componentes *hardware* y *software*.

2.3.1 Componentes hardware

Como se ha mencionado previamente, para conseguir un correcto funcionamiento del vehículo es necesario que estos recopilen información del entorno, siendo los componentes hardware los encargados de realizar esta tarea.

Dentro de estos componentes, cabe destacar: [9]

- **Ordenador central**: Tiene componente hardware y software. En él se procesa la información obtenida por los sensores del vehículo en forma de señales.
- **LIDAR** (*Laser Imaging Detection And Ranging*): Es un sistema de visión que, utilizando varios haces láser infrarrojos, indica la distancia a la que se encuentran los objetos en función del tiempo que tarda el haz en rebotar debido al objeto y en ser detectado por una lente (*Fig. 2.6)*. De esta forma, se obtiene una nube de puntos del exterior, conociéndose la posición y distancia de cada punto de una forma muy precisa. Es capaz de rotar sobre sí mismo, por lo que puede mapear el entorno en 360º. La principal ventaja de este sistema es que funciona bajo cualquier situación meteorológica, incluyendo condiciones extremas de iluminación. [10].

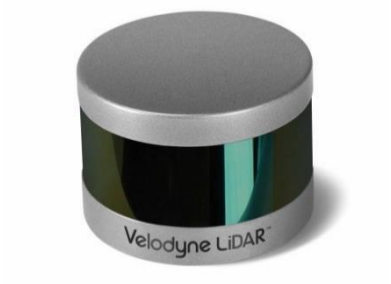


Fig. 2.6: Sensor LIDAR

- **GPS/IMU***:* El objetivo de estos dos sistemas es el de geolocalizar el vehículo. Por un lado, el GPS (*Global Positioning System*) permite determinar la posición de un objeto en la superficie terrestre, mientras que la IMU (*Inercial Measurement Unit*) permite controlar la velocidad, orientación y dirección de desplazamiento con la ayuda de acelerómetros y giróscopos. La IMU complementa la función del GPS, sirviéndole de ayuda en situaciones donde no hay cobertura suficiente para determinar la posición mediante satélites, ya que con la información de velocidad y dirección de desplazamiento del vehículo se consigue geolocalizar el vehículo en estas situaciones.

- **Cámaras de visión:** Los vehículos autónomos llevan implementadas varias cámaras de visión para reconocer los objetos del exterior, como personas, señales de tráfico, otros vehículos… Para ello, se emplean técnicas de visión artificial como las implementadas por Intel en las bibliotecas OpenCV.

- **Radares ultrasónicos:** Se tratan de radares que utilizan ultrasonidos para facilitar la maniobra de aparcamiento. Aunque este hardware no es exclusivo de este tipo de vehículos, es un requisito indispensable para la conducción autónoma.

- **Sensores y cámaras interiores**: El objetivo de estos sensores y cámaras interiores es el de conocer el estado del conductor en cualquier momento.

En la *Fig. 2.7* se puede observar un ejemplo de la ubicación de los distintos componentes hardware en un vehículo autónomo:

Fig. 2.7: Componentes hardware de los vehículos autónomos

2.3.2 Componentes software

Son los encargados de procesar toda la información recogida por los componentes hardware, y mandar las órdenes de actuación al vehículo.

Para ello, se utilizan distintas técnicas empleadas en la Inteligencia Artificial, como el aprendizaje automático (*machine learning*) y la visión artificial.

El denominado machine learning o aprendizaje automático consiste en la creación de programas que sean capaces de generalizar comportamientos en base a una información que se suministra mediante distintos ejemplos. En el caso de los vehículos autónomos, se le suministran un gran número de situaciones distintas con las que se puede encontrar circulando por la carretera. Con esta información, se realizan cálculos para generar decisiones y resultados fiables.

Por otra parte, la visión artificial se encarga de analizar el entorno mediante el uso de cámaras de visión instaladas previamente en el vehículo. [9]

La función que desempeña esta disciplina es muy importante, debido a que se encarga de la detección de los objetos del exterior, incluyendo vehículos, personas…

Existen numerosos softwares dedicados a la visión artificial, como MERLIC 3 y las ya mencionadas librerías OpenCV, teniendo estas últimas licencia BSD que permite su uso académico y comercial.

3. PLATAFORMA DE DESARROLLO Y HERRAMIENTAS UTILIZADAS

Este apartado tiene como objetivo explicar las herramientas utilizadas para llevar a cabo el proyecto y la plataforma de desarrollo en la que se ha implementado.

3.1 Linux

Como plataforma de desarrollo se ha utilizado el sistema operativo Linux, el cual se trata de un sistema operativo libre, es decir, puede ser utilizado, modificado y redistribuido de forma libre bajo términos GPL (*Licencia Pública General).*

El motivo principal por el cual se ha elegido Linux para este proyecto es por su compatibilidad con las herramientas que son necesarias en su elaboración.

Además, presenta una serie de ventajas que lo convierten en uno de los más utilizados en tareas de programación:

- ***Facilidad de uso:*** Linux se encuentra en numerosas distribuciones o *distros*, que incluyen paquetes de software predeterminados, como por ejemplo Ubuntu, la cual está orientada a usos generales. Esto facilita su utilización al estar instalado previamente el software. Además, incluye un Centro de Software desde el cual se pueden instalar distintos programas de una forma muy sencilla.
- ***Seguridad:*** Se trata del sistema operativo más utilizado en los servidores de alto rendimiento, debido a su alta seguridad.
- ***Libertad de uso:*** Se tiene acceso completo al propio Kernel, pudiendo configurar el sistema operativo a gusto propio.
- ***Eficiencia:*** Si se compara con Windows, Linux utiliza los recursos del ordenador de forma más eficiente, disminuyendo el consumo del ordenador y mejorando su rendimiento.

Como distribución, se ha elegido la versión 16.04.03 de 64 bits de Ubuntu, debido a que es la más utilizada actualmente y a su facilidad de uso.

Fig. 3.1 Icono característico de Linux

3.2 Lenguaje de programación

Para la realización de este proyecto se ha utilizado el lenguaje de programación C++.

Este lenguaje fue diseñado a mediados de los años 1980 por el científico danés Bjarne Stroustrup.

El objetivo de su creación fue extender el lenguaje de programación C para que fuera capaz de manipular objetos, por este motivo se denomina C++, que significa literalmente "incremento de C".

La principal razón de la elección de este lenguaje es su compatibilidad con las herramientas utilizadas en este proyecto, como *Point Cloud Library* y *OpenStreetMap.*

Las características principales de este lenguaje son:

- Sintaxis heredada del lenguaje C.
- Incorpora la programación orientada a objetos (*POO).*
- Permite la sobrecarga de operadores.
- Compatibilidad con lenguaje C.
- Permite la programación modular.
- Tiene un estándar ISO, siendo la última versión C++17 publicada oficialmente en ISO/IEC 14882:2017.
- Alta eficiencia y velocidad.
- Gran portabilidad, siendo soportado por varios sistemas operativos y compiladores.

3.3 Oracle VM VirtualBox

Debido a que el ordenador empleado en la realización de este proyecto tiene sistema operativo Windows, para poder utilizar Linux se ha hecho uso del software Oracle VM VirtualBox.

Se trata de un software de visualización, desarrollado actualmente por Oracle Corporation, que permite instalar sistemas operativos alternativos (denominados invitados) en otro sistema operativo (denominado anfitrión).

Con este software se tiene acceso total a la funcionalidad del sistema operativo sin necesidad de estar instalado en el ordenador.

La ventaja de utilizar una máquina virtual es la facilidad con la que puedes probar un sistema operativo distinto sin asumir ningún riesgo que afecte al sistema operativo principal. Sin embargo, los recursos del ordenador se verán compartidos entre ambos lo cual puede ralentizar su ejecución.

Fig. 3.2: Logotipo característico Oracle VM VirtualBox

3.4 OpenStreetMap

OpenStreetMap (*OSM)* es un proyecto colaborativo con el objetivo de crear mapas libres y editables de todo el mundo.

Estos mapas se crean usando información geográfica que ha sido previamente capturada por dispositivos GPS, y pueden ser modificados por cualquier usuario registrado.

En la actualidad, OpenStreetMap está expandiéndose de forma exponencial año tras año, gracias a la colaboración de todas las personas que crean y editan estos mapas. Para ilustrar este crecimiento, se muestra a continuación una gráfica (*Fig. 3.3)* proporcionada por la wiki de OpenStreetMap en la que se representan los valores acumulados de los nodos, vías y relaciones creadas, actualizada a fecha de 13/11/2017.

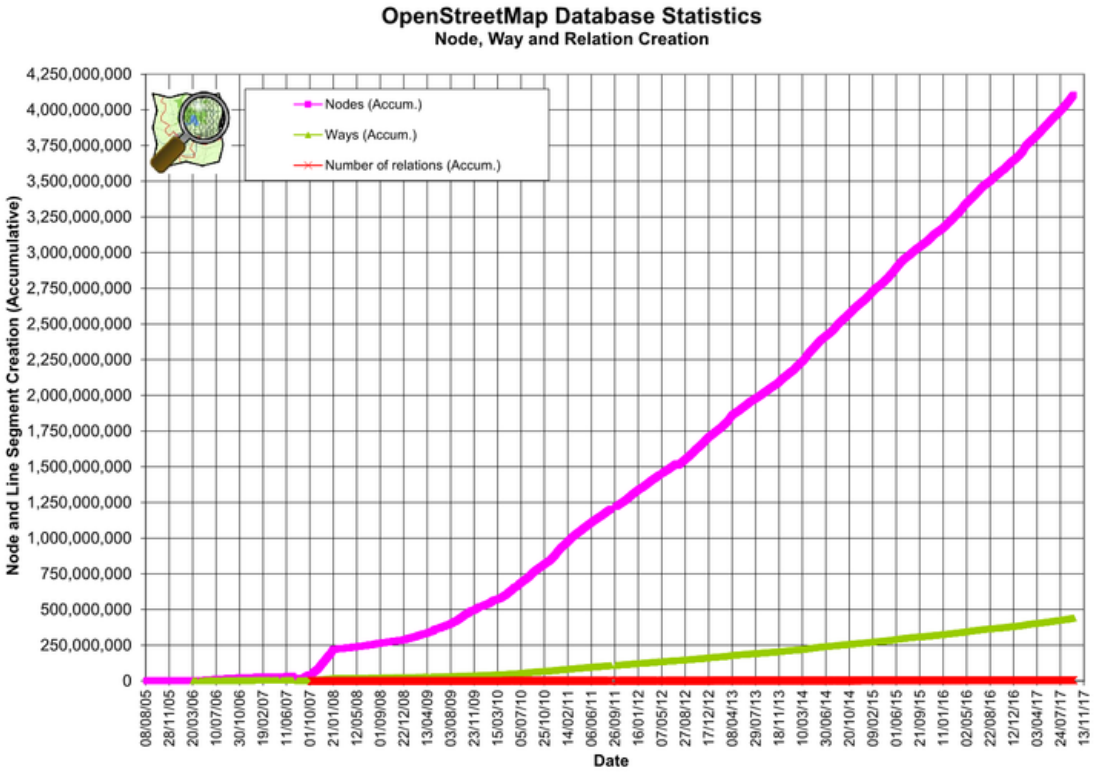


Fig. 3.3: Gráfica de valores acumulados de nodos, vías y relaciones en OpenStreetMap

Los mapas se componen de los siguientes elementos:

- *Nodos*: Se trata de un punto en el espacio, el cual viene definido por su latitud, longitud y un número denominado identificador. Además, pueden llevar asociados una etiqueta (*tag)*, que especifica la característica del nodo. Un ejemplo de la declaración de un nodo es mostrado en el siguiente código (*Fig. 3.4)*.

```
<node id="16375801" lat="48.9840938" lon="8.3962261" version="1">
    <tag k="source" v="LA-KA"/>
</node>
```

Fig. 3.4: Código de ejemplo de un nodo

- *Vías (ways)*: Son una agrupación de nodos con al menos una etiqueta o que pertenecen a una relación. Se puede distinguir entre vía cerrada o vía abierta, en función de si el primer nodo coincide con el último. En el siguiente código se muestra un ejemplo de una vía (*Fig. 3.5)*:

```
<way id="90458676" version="1">
    <nd ref="1049617993"/>
    <nd ref="2687101164"/>
    <nd ref="1049617953"/>
    <nd ref="1049618140"/>
    <nd ref="1049617993"/>
    <tag k="source" v="LA-KA"/>
    <tag k="building" v="yes"/>
    <tag k="addr:street" v="Neckarstraße"/>
    <tag k="addr:housenumber" v="47"/>
</way>
```

Fig. 3.5: Código ejemplo de una vía

En este ejemplo, la vía se compone de una agrupación de 5 nodos, caracterizados por su identificador (*ref)*, los cuales se corresponden con el edificio número 47 de la calle Neckarstraße, en la ciudad alemana de Karlsruhe.

- *Relaciones*: Están formadas por una agrupación de nodos y vías, y su función es definir relaciones geográficas entre los distintos elementos. En el siguiente código se muestra una relación entre dos vías (*Fig. 3.6):*

```
<relation id="6030015" version="1">
    <member type="way" ref="173816563" role=""/>
    <member type="way" ref="244951324" role=""/>
    <tag k="to" v="Karlsruhe Hauptbahnhof"/>
    <tag k="ref" v="S81"/>
    <tag k="via" v="Baiersbronn;Heselbach;Forbach (Schwarzwald);Rastatt"/>
    <tag k="from" v="Freudenstadt Hauptbahnhof"/>
    <tag k="line" v="light_rail"/>
    <tag k="name" v="Stadtbahn S81: Freudenstadt Hbf → Karlsruhe Hbf EILZUG ab Forbach"/>
    <tag k="type" v="route"/>
    <tag k="route" v="train"/>
    <tag k="colour" v="#6d682a"/>
    <tag k="network" v="Karlsruher Verkehrsverbund"/>
    <tag k="old_ref" v="S31"/>
    <tag k="operator" v="Albtal-Verkehrs-Gesellschaft mbH"/>
    <tag k="public_transport:version" v="2"/>
</relation>
```

Fig. 3.6: Código ejemplo de una relación

- *Etiquetas (tags)*: Pueden ser asignadas a cualquier elemento. Aportan información acerca de lo que representa cada nodo, vía y relación y se componen de una clave (*key)* y de un valor (*value).*

Fig. 3.7: Logotipo característico de OpenStreetMap

3.5 Libosmium

Para trabajar con los datos de los mapas descargados previamente, es necesario utilizar una librería.

En este caso, la librería elegida se llama *Libosmium*, la cual está desarrollada en los lenguajes C++ y Python.

Libosmium está regulada bajo los términos de licencia Attribution-ShareAlike License version 4.0.

Con el uso de esta librería se ha programado el código necesario para el procesamiento de los datos descargados de OpenStreetMap.

3.6 Point Cloud Library (*PCL)*

Point Cloud Library es un proyecto de código abierto a gran escala para el procesamiento de imágenes en dos y tres dimensiones mediante nubes de puntos.

Este proyecto está desarrollado bajo licencia BSD, siendo totalmente gratuito para usos comerciales y de investigación.

Su origen tuvo lugar en marzo de 2011 de la mano de Radu B. Rusu, y actualmente está siendo desarrollado por un elevado número de ingenieros y científicos de organizaciones de todo el mundo, incluyendo a la Universidad Carlos III de Madrid.

PCL contiene numerosos algoritmos enfocados en el estado del arte actual de la visión artificial y del procesamiento de imágenes, como algoritmos de segmentación, filtrado, estimación de superficies… etc.

Otra de las características de este proyecto es que es multiplataforma, posibilitando su uso en Windows, Linux, MacOS, iOS y Android.

Además, se encuentra dividido en una serie de librerías de código más pequeñas, que pueden ser compiladas individualmente, lo cual optimiza el coste computacional.

En la *Fig. 3.8* se muestran los módulos más importantes que componen PCL:

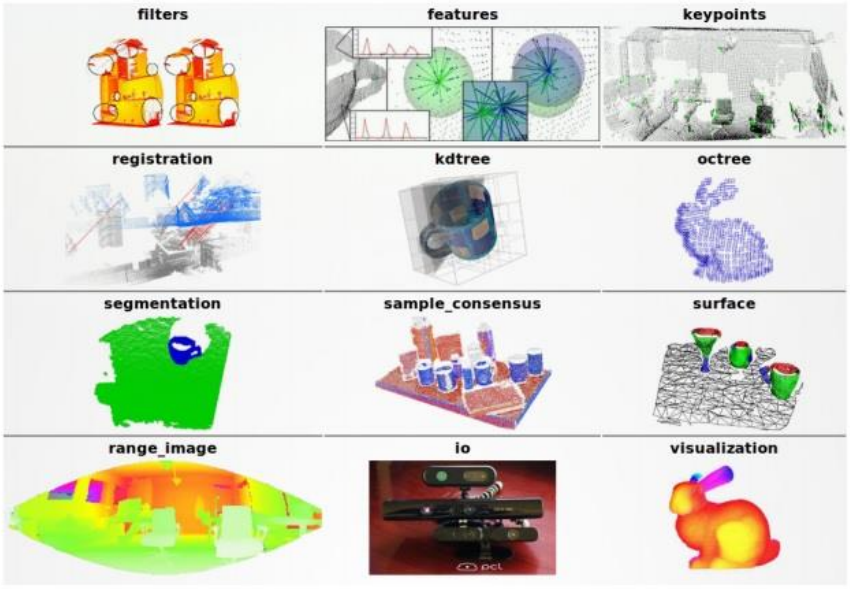


Fig. 3.8: Representación de los módulos más importantes de Point Cloud Library

Esta herramienta se va a utilizar con distintos objetivos, como por ejemplo guardar archivos de imagen en formato de nube de puntos PCD (*Point Cloud Data).*

Para ello es necesario convertir las imágenes digitales en nubes de puntos, lo cual se puede conseguir con el uso de cámaras estéreos y escáneres 3D. Además, con la inclusión de colores la nube de puntos se convierte en 4D, como se puede observar en la *Fig. 3.9:*

Fig. 3.9: Representación mediante nubes de puntos de distintos objetos incluyendo sus colores

Con los datos convertidos a formato PCD se pueden realizar numerosas funciones. La más importante utilizada en este proyecto es la segmentación de una nube de puntos, mediante la cual se pueden detectar distintos objetos o formas geométricas.

PCL proporciona distintos métodos para realizar esta segmentación, describiendo a continuación los más destacados:

- ***Kdtree**:* Es la abreviatura de *k-dimensional tree.* Es una estructura de datos que organiza los puntos en un espacio de *k* dimensiones.
  Este método utiliza únicamente planos perpendiculares a uno de los ejes del sistema de coordenadas. Una de las aplicaciones más comunes de este método es la buscar el punto vecino más cercano al punto que se está estudiando, en la cual se particiona el espacio en áreas cada vez más pequeñas como se muestra en la *Fig. 3.10:*

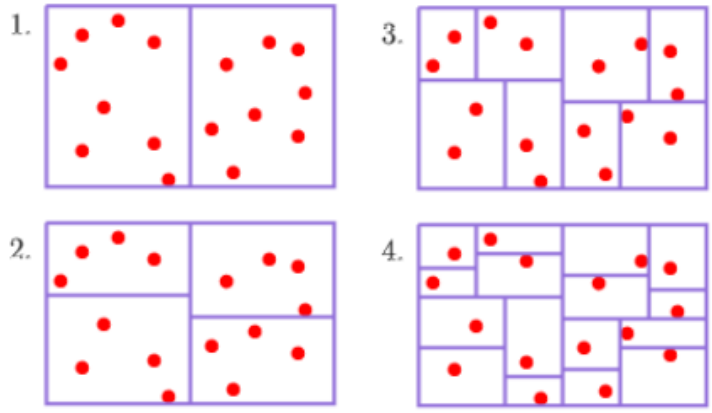


Fig. 3.10: Ejemplo utilización Kdtree

- ***Octree:*** Es una estructura de datos en forma de árbol, en la que cada nodo interno tiene 8 ramas. El uso más común de estas estructuras es el de particionar los espacios tridimensionales en ocho octantes. Un ejemplo de la utilización de este método se muestra en la *Fig. 3.11*:

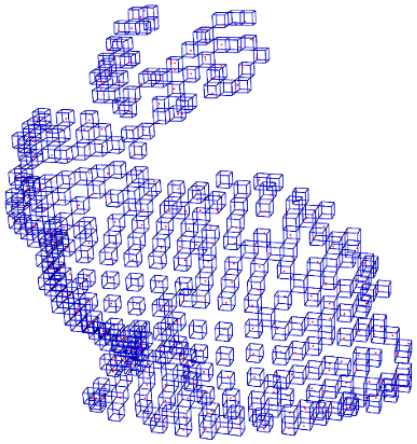


Fig. 3.11: Ejemplo utilización Octree

- ***Euclidean Cluster Extraction:*** El fundamento de este método es el de dividir la nube de puntos en partes más pequeñas para reducir el tiempo de procesamiento de forma significante.

  El algoritmo empleado incluye el método *Kdtree*, mediante el cual se realiza la búsqueda de los vecinos más cercanos.

  En la *Fig. 3.12* se muestra el resultado obtenido al aplicar este método a una nube de puntos que contiene diferentes objetos:

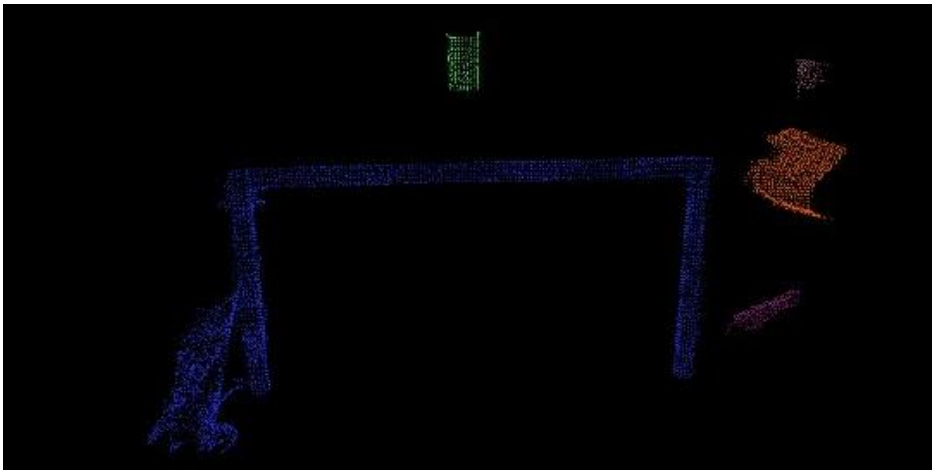


Fig. 3.12: Ejemplo utilización Euclidean Cluster Extraction

- ***Random Sample Consensus:*** Para este proyecto, el método utilizado es RANdom SAmple Consensus (*RANSAC*), el cual permite detectar distintos objetos que tengan una forma geométrica común, como esferas, planos, cilindros… El fundamento de este método es que todos los datos de una nube de puntos contienen *inliers* y *outliers* [11].

  Un punto es considerado como *inlier* si la distancia entre el punto original y el proporcionado por el modelo es inferior a un umbral (*threshold)*. Por otro lado, un punto es considerado *outlier* cuando está a una distancia mayor que ese umbral.

  La ventaja de utilizar este método de segmentación es que se trata de un proceso iterativo. Esto quiere decir que, una vez calculados los *inliers* y los *outliers*, estos últimos son considerados como una nueva nube de puntos sobre la cual se vuelve a realizar el método de forma iterativa hasta conseguir los resultados deseados.

  Un ejemplo de este método es mostrado en la *Fig. 3.13*:

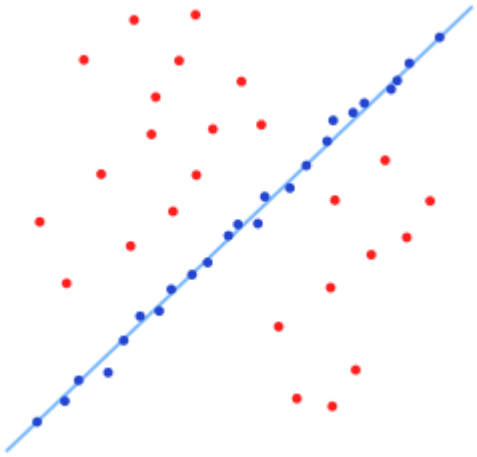


Fig. 3.13: Ejemplo utilización método RANSAC

En esta imagen se muestra un ejemplo de utilización de este método sobre una nube de puntos en dos dimensiones. La figura geométrica que se pretende segmentar es la de una línea, definida por los puntos de color azul, que se corresponden con los *inliers*, siendo los puntos rojos los *outliers*.

3.7 KITTI Vision Benchmark Suite

Como se ha mencionado en el apartado *Estado del arte*, el proyecto *KITTI Vision Benchmark* está enfocado en flujo estéreo y óptico y en odometría visual, entre otros.

Para la realización de este Trabajo de Fin de Grado se ha hecho uso de los datos correspondientes a la odometría visual de la ciudad de Karlsruhe disponibles en la página web del proyecto KITTI.

En concreto, se han utilizado los datos recogidos por el escáner láser Velodyne implantado en su vehículo, con el objetivo de guardarlos en formato PCD y poder analizarlos mediante nubes de puntos.

Estos datos están divididos en 22 secuencias estéreo, conteniendo las secuencias del 0 al 10 trayectorias de datos terrestres (comúnmente conocidos como *Ground Truth)* para poder comprobar los resultados que se obtengan en las distintas investigaciones.

Fig. 3.14: Ejemplos de secuencias que contienen datos terrestres (Ground Truth)

4. PROGRAMACIÓN Y DESARROLLO DEL PROYECTO

Una vez explicada la plataforma de desarrollo y las herramientas que se van a utilizar se procede a detallar los pasos seguidos en la elaboración de este proyecto:

4.1 Instalación de las herramientas necesarias

Para poder comenzar a desarrollar el proyecto es necesaria la instalación de las herramientas explicadas en el apartado anterior.

4.1.1 Instalación Libosmium

Como ya se ha mencionado, Libosmium es una librería de C++ que permite trabajar con los datos obtenidos de OpenStreetMap, tanto en el sistema operativo Linux como MacOS.

El primer paso a realizar es la instalación de las dependencias necesarias para el correcto funcionamiento de la librería, como *CMake, Make, Google Protocol Buffers, Protozero, Expat, ZLib y bz2lib.*

En la *Fig. 4.1* se muestra el código empleado para la instalación de todas las dependencias:

```
apt-get install -q -y \
    cmake \
    doxygen \
    g++ \
    git \
    graphviz \
    libboost-dev \
    libbz2-dev \
    libexpat1-dev \
    libgdal-dev \
    libgeos++-dev \
    libproj-dev \
    libsparsehash-dev \
    make \
    ruby \
    ruby-json \
    spatialite-bin \
    zlib1g-dev
```

Fig. 4.1: Instalación de las dependencias necesarias para Libosmium

Una vez instaladas las dependencias, es necesario clonar el código en la máquina virtual utilizada, el cual está disponible en [12] de forma totalmente gratuita.

A continuación, se crean los directorios en los cuales se va a almacenar el código clonado y se utiliza el comando *cmake* para crear una configuración inicial.

Por último, se utiliza el comando *make* que finaliza la instalación.

4.1.2 Instalación Point Cloud Library

Point Cloud Library está disponible para los sistemas operativos Windows, Linux y MacOS.

Para poder utilizar las funcionalidades de Point Cloud Library en Ubuntu es necesario instalarlo en la máquina virtual mediante vía PPA (*Personal Package Archives)*, la cual permite tener siempre actualizado el software de Point Cloud Library a la versión más reciente.

En la *Fig. 4.2* se muestra el código utilizado en la instalación para la versión Ubuntu de la máquina virtual:

```
sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
sudo apt-get update
sudo apt-get install libpcl-all
```

Fig. 4.2: Instalación Point Cloud Library en Ubuntu

4.2 Adquisición de los datos de OpenStreetMap

Para poder utilizar la librería *Libosmium* es necesario tener descargado el mapa de OpenStreetMap en uno de los formatos soportados por la librería (en este caso, el formato elegido es OSM).

Existen varias páginas web desde las cuales descargar estos mapas, siendo [13] la elegida debido a su sencillez a la hora de crear un área de extracción.

Como para este proyecto se utilizan los datos disponibles de la ciudad alemana Karlsruhe, se ha descargado una porción del mapa de dicha ciudad.

El motivo de descargar una parte del mapa en vez del de toda la ciudad es para agilizar la compilación y ejecución del programa.

En la *Fig. 4.3* se muestra el área descargada:

Fig. 4.3: Zona de Karlsruhe descargada

4.3 Pruebas iniciales con OpenStreetMap

Una vez instalada la librería *Libosmium* y descargado el mapa en formato OSM se puede empezar a desarrollar el proyecto.

El objetivo final que se pretende conseguir en este apartado es el de detectar los edificios cercanos a una ubicación dada y calcular la distancia entre sus paredes y el punto de referencia.

Para ello, se han realizado una serie de pruebas las cuales se detallan a continuación:

4.3.1 Detección de edificios en una zona determinada

El primer objetivo del programa es detectar los edificios de una zona dada una ubicación en latitud y longitud.

Por tanto, como primera prueba se programó un código que, a partir de una ubicación y un umbral determinado, recorriese todos los nodos del mapa y mostrase únicamente los detectados con su respectiva referencia.

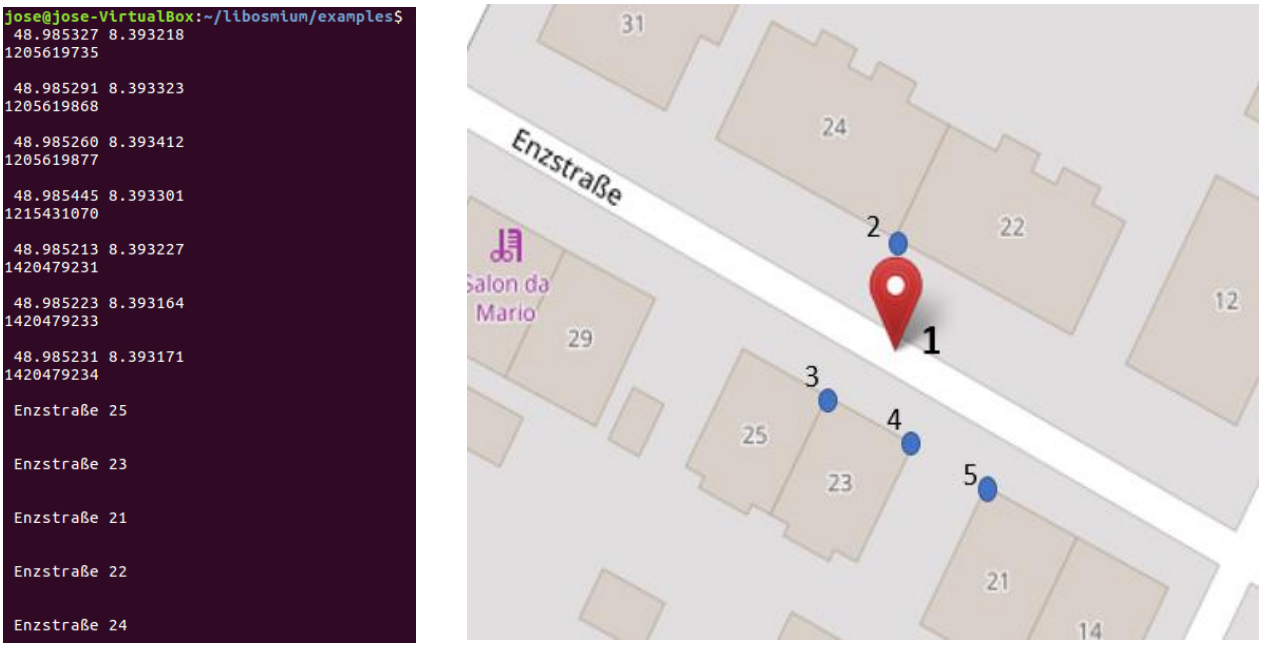
Después, se analiza cada camino (*way*), los cuales están compuestos por nodos como se ha mencionado anteriormente.

Para cada uno de ellos, si alguno de sus nodos coincide con al menos un nodo detectado se comprueba si ese camino corresponde o no a un edificio, mostrando por pantalla los edificios detectados en caso afirmativo.

OpenStreetMap clasifica los edificios bajo la etiqueta *building*, por lo que en el código programado se van a buscar todos los caminos que contengan esa etiqueta.

En esta primera prueba la ubicación elegida es 48.9853600 grados de latitud y 8.3933011 grados de longitud, y el umbral 0.00015 grados.

En las *Fig. 4.4* se muestra el resultado proporcionado por el programa tanto en consola como en el mapa:



(a) *Resultado proporcionado por consola* (b) *Resultado mostrado en el mapa*

Fig. 4.4: Resultado detección de edificios

En la *Fig. 4.4 (b)*, el número 1 se corresponde con la ubicación elegida y los demás números son los nodos detectados que están contenidos en los edificios, mientras que en la de

la izquierda se muestran primero los nodos detectados en latitud y longitud junto con su referencia y después los edificios.

Como se puede comprobar, lo obtenido por el programa se corresponde con la ubicación de los edificios en el mapa.

A partir de este resultado se puede ir calibrando la búsqueda mediante la modificación del valor del umbral, para que realice la comprobación en un área mayor o menor.

Una vez realizada esta primera prueba inicial, el siguiente paso fue comprobar que el código funcionaba en cualquier zona del mapa, por lo que se realizaron experimentos en distintos puntos.

Aunque el código funcionaba en la mayoría de los casos, existían algunos nodos que, a pesar de estar incluidos en un edificio, el programa no los detectaba.

Esto se debe a que OpenStreetMap no engloba a cualquier edificio bajo la etiqueta *building*, sino que edificios como colegios, hospitales, iglesias… etc. tienen su propia etiqueta.

Por tanto, fue necesario actualizar el código para que también tuviera en cuenta otras etiquetas, y se comprobó que los resultados eran correctos en cualquier punto.

4.3.2 Cálculo de distancias a los edificios detectados

Después de haber detectado los edificios, el siguiente paso a seguir es calcular la distancia que hay entre las paredes de estos y el punto de referencia.

Debido a que se está trabajando con latitudes y longitudes, para poder calcular la distancia entre dos puntos primero hay que convertir estas magnitudes en dos ejes con los cuales se pueda operar (*X* e *Y).*

En el caso de la latitud, la equivalencia es de 111,325 km por cada grado. Sin embargo, la equivalencia de la longitud depende la latitud en la que se encuentre cada punto de la siguiente forma:

$$longitud = \cos(latitud) \cdot 40.076 \qquad (4.1)$$

Donde, *longitud* se refiere a la circunferencia del paralelo que se encuentra a esa latitud, y *40.076* es la longitud de la circunferencia del ecuador en kilómetros.

Por tanto, si se desea conocer la distancia que corresponde a un grado de longitud, hay que dividir la ecuación por 360, resultando:

$$°longitud = \cos(latitud) \cdot 111{,}325 \qquad (4.2)$$

En este caso, el eje Y se va a corresponder con la *latitud* y el eje X con la *longitud*, ambas medidas en kilómetros para poder trabajar con ellas.

En un primer momento, para simplificar la programación, se calculó la distancia entre el punto de referencia y cada nodo detectado.

Sin embargo, esto no proporciona resultados correctos ya que la distancia obtenida no se corresponde con la distancia en perpendicular, por lo que fue necesario calcular las rectas que forman el polígono de cada edificio y hallar la distancia en perpendicular desde el punto de referencia hasta esas rectas.

Como OpenStreetMap no proporciona estas rectas, se tuvieron que calcular utilizando geometría analítica, al igual que la distancia, por lo que se han implementado una serie de funciones que realizan estos cálculos.

El procedimiento seguido es el siguiente:

1. Para cada edificio detectado, se calcula la equivalencia en kilómetros de cada nodo.
2. Se hallan las rectas de los edificios, recorriendo nodo a nodo el polígono que lo forma.
3. Se calcula la distancia en perpendicular que hay entre cada recta y el punto de referencia, el cual también debe ser convertido a kilómetros.
4. Se muestra por pantalla el resultado obtenido.

Para esta segunda prueba, el punto de referencia considerado es el mismo que en la anterior, es decir, 48.9853600 grados de latitud y 8.3933011 grados de longitud, utilizando el mismo umbral (0.00015 grados).

En las *Fig. 4.5* se muestra el resultado que proporciona el programa y su representación en el mapa:

(a) *Resultado proporcionado por consola* *(b) Resultado mostrado en el mapa*

Fig. 4.5: Resultado del cálculo de distancias a edificios

En la *Fig. 4.5 (a)* se muestra el resultado que proporciona el código por pantalla. En este caso, se ha elegido analizar el edificio número 21 debido a que es rectangular y facilita el entendimiento de la imagen de la derecha.

Lo primero que se muestra por pantalla es el nombre del edificio. Después, se muestra la recta y por último su distancia en perpendicular.

En la *Fig. 4.5 (b)* se representa el resultado en el mapa, siendo el punto P el punto de referencia, y las distintas rectas etiquetadas con el número de orden en el que aparecen en la consola.

Como se puede comprobar, las rectas 1 y 3 tienen el mismo vector director, al igual que las rectas 2 y 4, lo cual indica que son paralelas dos a dos y que se corresponde con el dibujo del mapa.

Además, si se realiza el producto escalar de los vectores directores de estas rectas se comprueba que son perpendiculares, es decir, que su producto escalar es 0. Utilizando las dos primeras rectas como ejemplo:

$$Producto\ Escalar = (-0{,}515688 \cdot 1{,}93922) + \big(-1 \cdot (-1)\big) = 0{,}00032 \qquad (4.3)$$

Respecto a los valores de distancia calculados, la recta 3 es la que más cerca está del punto **P**. Esto tiene sentido ya que se puede apreciar a simple vista que es la

recta más cercana en perpendicular al punto, por lo que se puede concluir que el código es correcto.

Para comprobar si el código funcionaba en cualquier zona, se realizaron experimentos en distintas partes de la ciudad, comprobando finalmente que los resultados eran correctos.

4.4 Pruebas iniciales con Point Cloud Library

El objetivo que se pretende conseguir mediante la utilización de esta herramienta es la detección de los distintos planos presentes en una nube de puntos para más tarde relacionar esos planos con los edificios detectados en el apartado anterior.

Como se ha mencionado en el apartado 3.5, en este proyecto se ha utilizado el método RANSAC para realizar la segmentación.

El motivo de haber elegido este método en vez de las otras opciones disponibles es que permite segmentar la nube de puntos en formas geométricas comunes como planos, cilindros y esferas, obteniendo resultados muy precisos.

Para comprobar el funcionamiento de este método se realizó una prueba inicial, en la que se creó una nube de 1200 puntos.

Estos puntos se repartieron en 3 planos de 400 puntos cada uno, siendo los planos los siguientes:

1. $z = 2$ (4.4)
2. $y = 2$ (4.5)
3. $z = 3$ (4.6)

Este método requiere de la especificación de distintos parámetros. El primero de ellos es el eje de búsqueda, es decir, el eje en el cual el algoritmo segmentará la nube de puntos. El segundo es un umbral de ángulo y el tercero un umbral de distancia.

Para esta primera prueba inicial, el eje de búsqueda es el (0, 1, 0), el cual se corresponde con el eje Y. Por tanto, el único plano que debería detectar es el segundo. Respecto a los otros dos parámetros, el umbral de ángulo especificado es de 10 grados y de distancia 20 cm, para asegurarnos que únicamente detecta ese plano y no los demás.

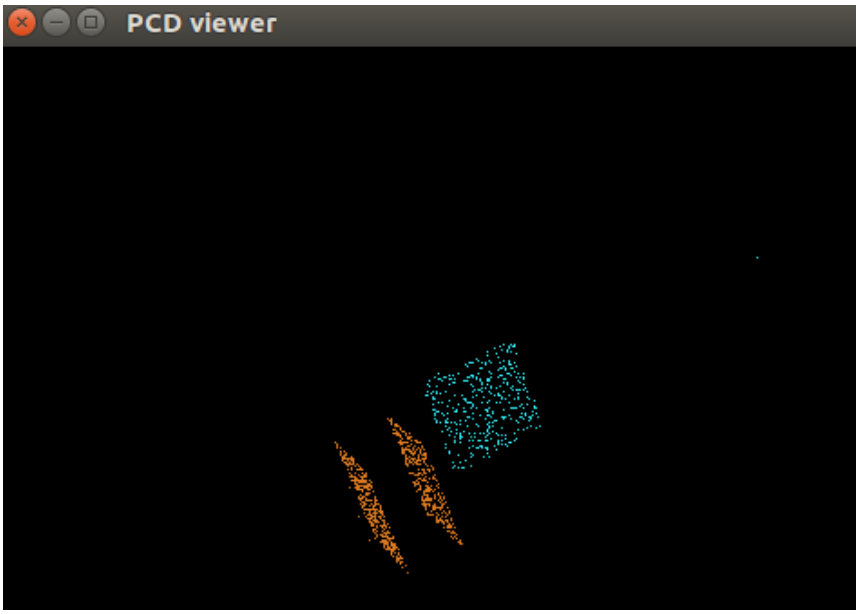


Fig. 4.6: Resultado prueba inicial PCL

En la *Fig. 4.6* se observa el resultado de la segmentación utilizando el visor de Point Cloud. En ella se representan tanto los *inliers* como los *outliers* obtenidos, siendo estos últimos los de color naranja y de color azul los primeros. En este caso, los *inliers* se corresponden con los puntos que forman el plano a buscar, es decir, el plano $y = 2$ tal como era de esperar.

Por tanto, se puede concluir que el resultado es el esperado y que el método funciona correctamente.

4.4.1 Adquisición y carga de ficheros binarios

Una vez comprobado el funcionamiento de la segmentación, el siguiente paso es cargar los ficheros binarios que contienen las nubes de puntos.

Estos ficheros están disponibles de forma gratuita en [14] correspondiente al proyecto KITTI Vision Benchmark Suite, mencionado en apartados anteriores.

Están divididos en 22 secuencias estéreo, repartidas a lo largo de la ciudad alemana Karlsruhe, siendo la secuencia número 7 la utilizada en este proyecto debido a que se corresponde con una zona urbana con numerosos edificios y, además, contiene su propio *Ground Truth* para poder comprobar los resultados obtenidos.

En la *Fig. 4.7* se muestra el recorrido del vehículo en la secuencia número 7:

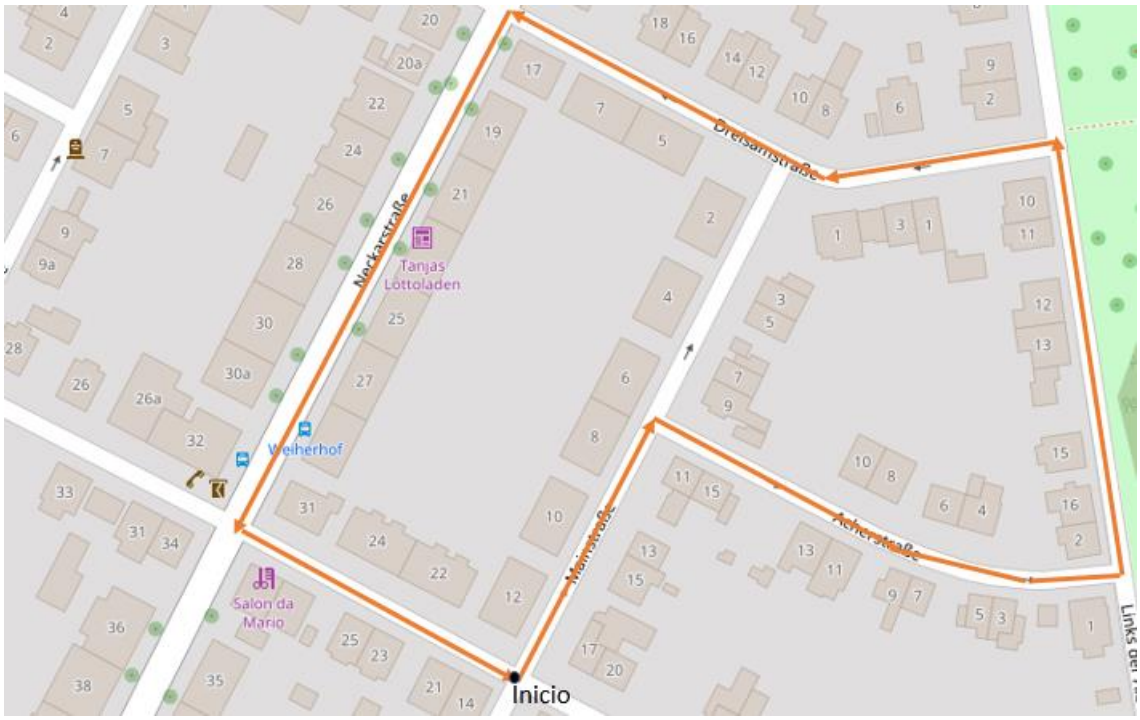


Fig. 4.7: Recorrido de la secuencia número 7

Además de estos ficheros binarios, también se dispone de las imágenes del recorrido del vehículo en cada secuencia, para saber a qué zona del mapa corresponde cada fichero.

Por tanto, teniendo descargados todos estos datos, se procede a programar un código que lea un fichero binario y lo guarde en una nube de puntos.

Como primer ejemplo, se ha utilizado el fichero correspondiente a la imagen número 3, la cual se corresponde con el punto de Inicio de la figura anterior, obteniendo el siguiente resultado (*Fig. 4.8)*:

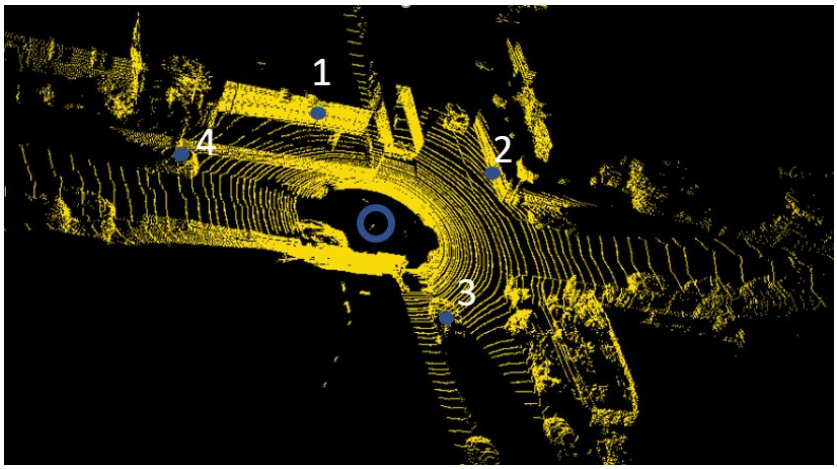


Fig. 4.8: Nube de puntos correspondiente a la imagen número 3

En la *Fig. 4.8*, el lugar donde está ubicado el láser del vehículo es el círculo azul. Como se puede observar, alrededor del punto azul no se detectan puntos ya que se corresponde con la superficie que tapa el propio vehículo.

El punto número 1 indica la superficie de una pared vertical, que se corresponde con la del edificio ubicado en esa zona del mapa, paralelo a la carretera.

Por otro lado, el punto número 2 se trata de la pared de un edificio en perpendicular a la carretera en la que está ubicado el vehículo. Por tanto, esto indica que la imagen se trata de una intersección de dos calles.

Otros conjuntos de puntos detectados son los vehículos, como el que indica el punto 3. Además, también se detectan otros elementos del entorno como árboles (punto 4).

4.4.2 Filtrado de los puntos del suelo

A pesar de haber procesado correctamente los ficheros binarios para poder tratarlos como una nube de puntos, se puede observar en la imagen del apartado anterior que existen numerosos puntos correspondientes al plano del suelo que dificultan la visualización y no son necesarios para este proyecto.

Por tanto, mediante el método de segmentación empleado en los apartados anteriores se procede a filtrar estos puntos para obtener una imagen más fácil de visualizar.

En este caso, el plano del suelo se corresponde con el eje Z, por lo que la condición del eje a buscar es el (0, 0, 1).

Fig.4.9: Filtrado de los puntos del suelo

Como se puede comprobar en la *Fig. 4.9*, los puntos del suelo han sido eliminados obteniendo una imagen mucho más fácil de visualizar y analizar. Además, se han representado los ejes que emplea el visualizador de Point Cloud.

Al utilizar este método de segmentación se pueden obtener los coeficientes del plano detectado, lo cual es útil para comprobar si el resultado es correcto o no. En este caso, los coeficientes obtenidos han sido los siguientes:

$$-0{,}016x + 0{,}006y + 0{,}999z = -1{,}73 \tag{4.7}$$

Analizando estos coeficientes se puede concluir que el plano detectado es el esperado, ya que se corresponde con un plano del eje Z.

El término independiente indica la distancia a la que está el plano, medida desde el láser. Todas las medidas de distancias proporcionadas por este método son expresadas en metros.

Por tanto, este resultado indica que el suelo está a 1,73 metros por debajo del láser, lo cual es correcto.

4.4.3 Segmentación de planos paralelos

El siguiente paso es segmentar la nube de puntos para obtener los planos paralelos y perpendiculares a la carretera.

En este primer caso se van a segmentar únicamente los planos paralelos, es decir, los planos del eje X, tal y como están representados los ejes en el apartado anterior.

El método utilizado es el mismo que en los anteriores apartados, pero hay que realizar una serie de cambios.

El primer cambio con respecto a los otros ejemplos es que hay que programar un bucle que detecte estos planos, ya que el método de segmentación detecta únicamente un plano por cada iteración.

La segunda diferencia es debida al tamaño de la nube de puntos del plano detectado. Este método detecta nubes de puntos que forman una forma geométrica

parecida a la predefinida, pero sin tener en cuenta el número de puntos totales. Por tanto, habrá planos detectados que sean formados por un número muy pequeño de puntos, los cuales no se corresponden con la pared de ningún edificio, pero sí con un vehículo, un cartel… La solución a este problema es imponer una condición en el bucle de que el tamaño de la nube de puntos detectada sea mayor de un número, en este caso, 3000 puntos.

Teniendo en cuenta estas modificaciones, el resultado obtenido es el siguiente (*Fig. 4.10)*:

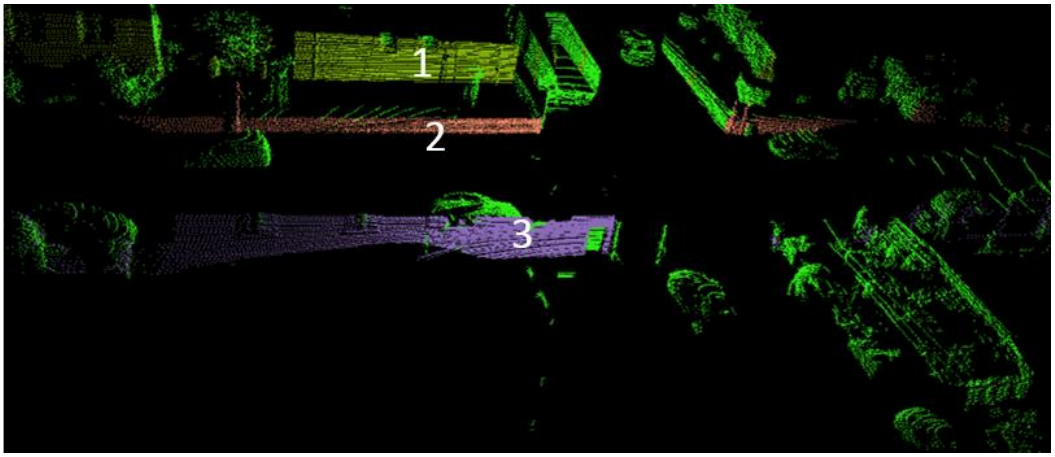


Fig.4.10: Segmentación de planos paralelos

En la *Fig. 4.10* se observa el resultado obtenido después de realizar la segmentación. El número 1 se corresponde con la pared del edificio que está a la izquierda del vehículo; el número 2 es el plano que forman las vallas de madera del edificio, y el número 3 es la pared del edificio que está a la derecha del vehículo.

A continuación, se muestran los coeficientes obtenidos de los 3 planos:

$$1.\ 0{,}9982x + 0{,}0059y + 0{,}008z = -10{,}07 \tag{4.8}$$

$$2.\ 0{,}9981x + 0{,}005y + 0{,}0019z = -5{,}11 \tag{4.9}$$

$$3.\ 0.9962x + 0{,}0067y + 0{,}00612z = 4{,}867 \tag{4.10}$$

Como se puede observar, los tres planos son pertenecientes al eje X, con valores muy pequeños en los demás ejes que se pueden aproximar a cero.

Además, los valores de las distancias obtenidos son los esperados, ya que el plano número 1 es el más alejado (10,07 metros), y, junto con el plano número 2, tienen valores de distancia negativos al estar en la dirección negativa del eje X.

4.4.4 Segmentación de planos perpendiculares

Realizado el proceso, se procede a segmentar los planos perpendiculares a la carretera, es decir, los planos del eje Y.

Al igual que en el apartado anterior, hay que realizar los mismos cambios para poder segmentar en bucle.

Sin embargo, en el caso de este tipo de planos el tamaño de la nube de puntos obtenida es notablemente menor al de los planos paralelos, debido al alcance del láser en esta dirección. Por tanto, se impone como condición un tamaño de 1000 puntos en vez de los 3000 del apartado anterior.

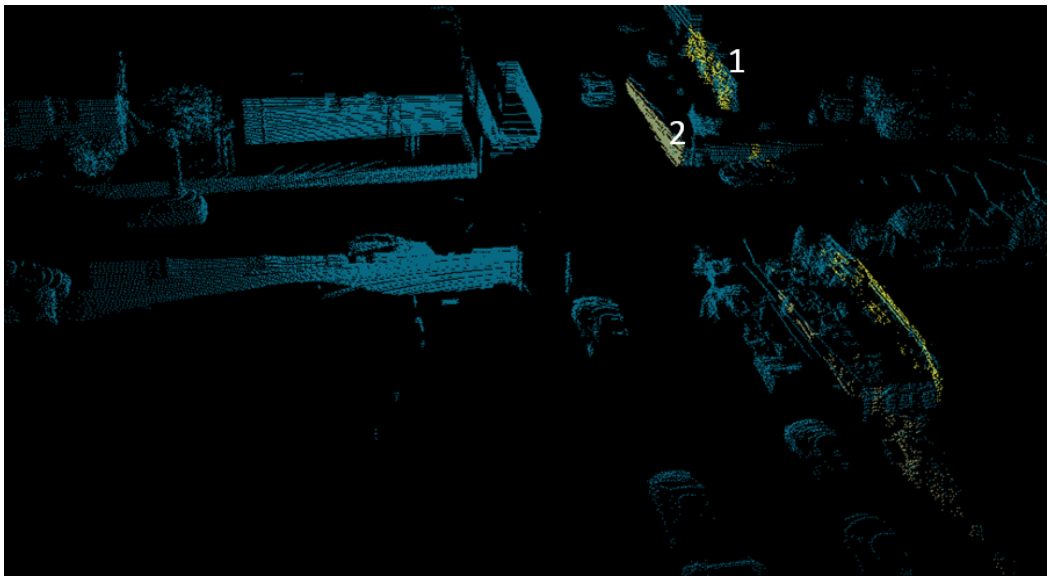


Fig.4.11: Segmentación de planos perpendiculares

La *Fig. 4.11* muestra el resultado obtenido después de la segmentación. En ella, se resaltan dos planos. El primero de ellos se corresponde con la fachada del edificio que está en perpendicular a la carretera del vehículo, y el segundo se identifica con unas vallas de madera que están en frente del edificio.

Al haber disminuido el número de puntos mínimo, se obtienen muchos más planos que en el apartado anterior. Sin embargo, estos planos no son válidos ya que no

representan ninguna parte importante del entorno, por lo que no se han mostrado en la imagen.

Los coeficientes obtenidos para estos dos planos son los siguientes:

1. $0{,}00302x + 0{,}99871y + 0{,}004z = 14{,}47$ (4.11)

2. $0{,}009x + 0{,}9956y + 0{,}002z = 10{,}83$ (4.12)

Como se puede observar, ambos planos se corresponden con un plano del eje Y, con pequeños errores en los demás coeficientes que se pueden considerar cero.

Respecto a los términos independientes, el del primer plano es mayor que el del segundo, lo cual es lógico ya que está más alejado del vehículo.

En conclusión, la segmentación realizada de ambos tipos de planos es correcta.

4.5 Unión OpenStreetMap y Point Cloud Library

Después de comprobar el funcionamiento de las segmentaciones, el siguiente paso es unir OpenStreetMap y Point Cloud Library en un mismo programa para poder comparar ambos resultados y verificar que los planos obtenidos se corresponden con edificios.

Respecto a OpenStreetMap, es necesario modificar el código realizado en los apartados anteriores para que distinga entre paredes perpendiculares y paralelas a la carretera. Para ello, se utilizan dos puntos de la carretera y se crea el vector que los contiene, y para cada pared detectada se comprueba el ángulo que forman el vector de la carretera y el de la pared.

Si ese ángulo es 0 ó 180 grados, se considera como una pared paralela a la carretera y si es 90 ó 270 grados, como perpendicular.

Por tanto, se guardan todos los datos de OpenStreetMap en distintos vectores, tanto para las paredes paralelas como para las perpendiculares. Uno de ellos guardará la dirección (calle y número), otro se encargará de almacenar todas las rectas que forman los edificios y el último vector contendrá las distancias desde cada recta al punto de referencia.

Lo mismo hay que hacer con los datos de Point Cloud. Es necesario almacenar todos los planos detectados, por un lado los paralelos en un vector y los perpendiculares en otros.

Finalmente, se juntan los dos programas en uno y se procede a completar el código como se explica en los siguientes apartados.

5. EXPERIMENTOS

En este apartado se explican los experimentos realizados para completar el proyecto, una vez terminadas las pruebas iniciales en OpenStreetMap y Point Cloud Library.

5.1 Experimento 1: Comparación de distancias

El objetivo del primer experimento es comparar las distancias obtenidas por ambas herramientas, distinguiendo entre planos paralelos y planos perpendiculares.

Para ello, se compara el valor del término independiente de cada plano detectado (*distancia*) con todos los valores de distancias obtenidos mediante OpenStreetMap.

Si ambos valores coinciden teniendo en cuenta un margen de error, en este caso de 2 metros, se muestra por pantalla la dirección del edificio y ambas distancias.

El punto de referencia considerado en este ejemplo es el 48.985644, 8.394088, mostrado en la *Fig. 5.1:*

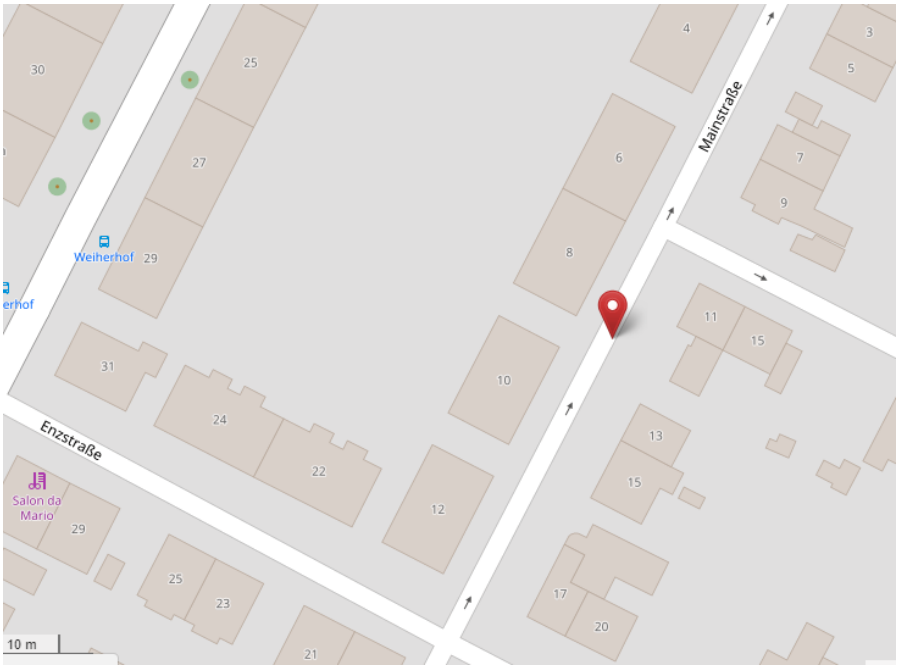


Fig. 5.1: Punto de referencia experimento 1

Aplicando el programa a este punto de referencia, el resultado obtenido es el siguiente (*Fig. 5.2)*:

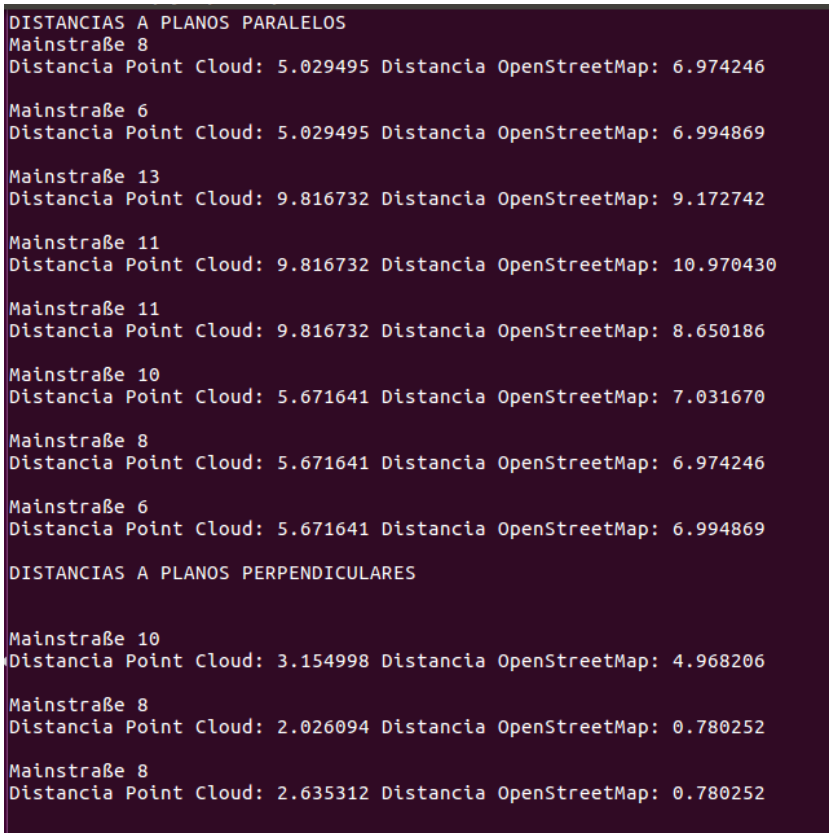


Fig. 5.2: Comparación de distancias OSM y PCL

En este ejemplo, primero se muestran los planos paralelos cuyas distancias coinciden con las paredes paralelas de los edificios de OSM, teniendo en cuenta el umbral (margen de error).

En el caso de los planos perpendiculares, en la imagen no se muestran todos los obtenidos ya que son un número muy elevado.

Como se puede comprobar, los edificios detectados son los más cercanos al punto de referencia.

Si se modifica el umbral se puede ir observando cómo coinciden más o menos edificios en función de si se incrementa o decrementa, pudiendo calibrar el

funcionamiento del programa. Por ejemplo, si el margen de error es de 1 metro se obtiene el siguiente resultado (*Fig. 5.3)*:

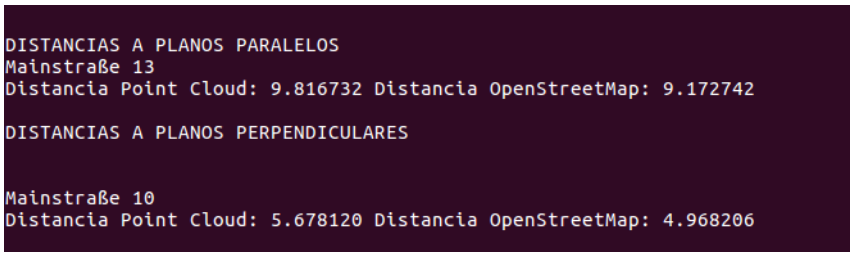


Fig. 5.3: Comparación de distancias OSM y PCL con umbral de 1 metro

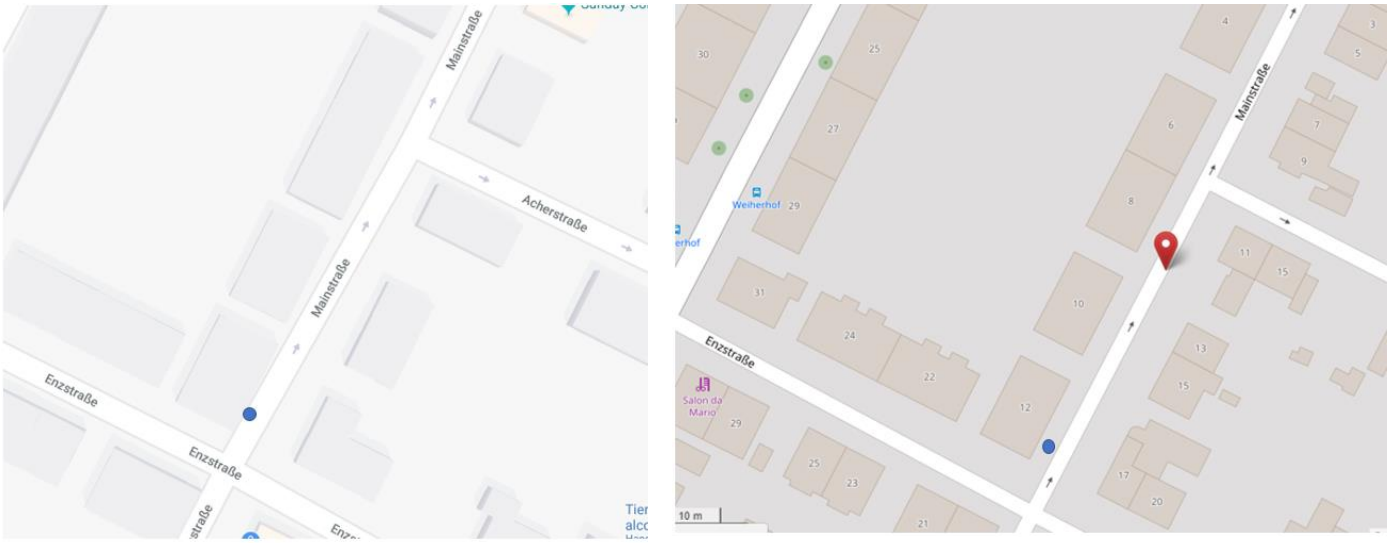
En este caso, el número de planos detectados es inferior, ya que la restricción de distancias ha cambiado, obteniendo resultados más precisos.

Sin embargo, esto indica que la mayoría de los planos tienen un error superior a 2 metros en la comparación de distancias.

Se realizaron pruebas con distintos puntos del mapa para comprobar si el error disminuía, pero se obtuvieron los mismos resultados e incluso errores más altos.

Después de analizar el origen de este error, se llegó a la conclusión de que es debido a los datos proporcionados por OpenStreetMap, ya que la ubicación real de los edificios no siempre es la que proporciona esta herramienta.

Para poder verlo con claridad, se muestra a continuación una comparación del mapa de Google Maps y de OpenStreetMap (*Fig. 5.4)*:



(a) *Mapa de Google Maps* *(b) Mapa de OpenStreetMap*

Fig. 5.4: Comparación Google Maps y OpenStreetMap

Como se puede observar, en la *Fig. 5.4 (a)* el edificio ubicado en el punto azul está mucho más próximo a la carretera de lo que se muestra en el mapa de OpenStreetMap. Esta diferencia es el origen del error obtenido al comparar las distancias de *OSM* y *PCL*, por lo que para solucionarlo es necesario aumentar el margen de error para que se tenga en cuenta este hecho.

5.2 Experimento 2: Determinación de la ubicación del vehículo

En este segundo experimento, el objetivo es determinar la ubicación del vehículo uniendo los resultados de las dos herramientas y utilizando las distancias calculadas en el experimento anterior.

Para ello, el método seguido es el siguiente:

- Para cada recta que cumpla las condiciones del margen de error de distancia se crea una recta paralela a ella a la distancia proporcionada por Point Cloud Library.

- Se guardan en un vector las rectas que sean paralelas a la carretera y en otro las que sean perpendiculares, ambas desplazadas la distancia correspondiente.

- Se calcula el punto de corte de cada recta paralela con todas las perpendiculares, y se guarda por un lado el término del eje X y por otro el del eje Y del punto calculado.

- Se calcula la distancia entre el punto de referencia y cada punto de corte calculada en el paso anterior, denominada *offset experimental*.

- Se realizan tres comprobaciones para cada punto de corte calculado:

    1. El offset experimental debe ser igual a la distancia que existe entre el punto de referencia y el punto donde está ubicado el coche (*offset)*, con un margen de error (umbral).

    2. El término X del punto de corte debe ser igual al término X del punto de ubicación del vehículo, también con un margen de error.

    3. Al igual que con el término X, se realiza la misma comprobación para el Y.

- Se calcula el error de *offset* cometido, es decir, el error que existe entre el offset y el experimental, para cada punto que cumpla las tres comprobaciones.

- Por último, el punto de corte calculado será el resultante de realizar la media entre todos los términos X e Y, transformándolos en coordenadas latitud y longitud teniendo en cuenta que la X se corresponde con la longitud y la Y con la latitud. El error de offset se calculará como la media de todos los errores de los puntos aceptables.

En la *Fig. 5.5* se muestra una representación del método, utilizando únicamente una recta paralela y una perpendicular y determinando su punto de corte de forma manual:

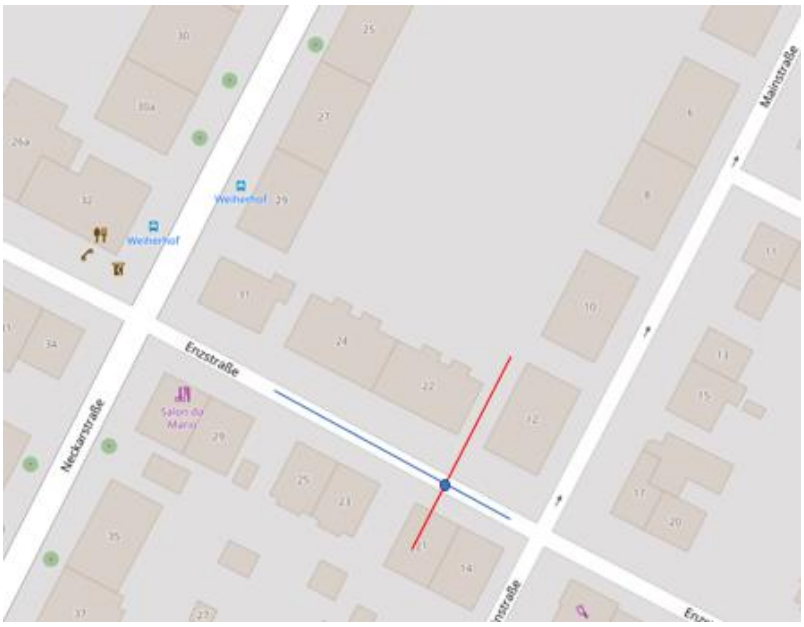


Fig. 5.5: Ejemplo de la determinación de la ubicación

Para poder conocer el punto donde está ubicado el vehículo se dispone de los datos contenidos en el archivo de texto *Ground Truth*.

Este archivo de texto está incluido en cada secuencia de imágenes, utilizando en este apartado el correspondiente a la secuencia número 7.

El *Ground Truth* está compuesto por tantas matrices como imágenes haya en la secuencia (1101 en este caso), con la siguiente estructura:

$$(\boldsymbol{R}|\boldsymbol{T})$$

Donde R se corresponde con una matriz de rotación 3x3 y T con un vector de traslación 3x1.

Para este proyecto sólo se va a tener en cuenta el vector de traslación. Este vector contiene la distancia que hay entre el primer punto de la secuencia (punto de referencia) y el punto de la imagen donde se esté analizando, para cada uno de los ejes (X, Y, Z).

Sin embargo, estos ejes no se corresponden con los ejes de latitud y longitud utilizados en los anteriores apartados, por lo que es necesario convertir los ejes del vector de traslación para poder calcular la posición del vehículo en cada imagen.

En la *Fig. 5.6* se representan ambos sistemas de ejes:

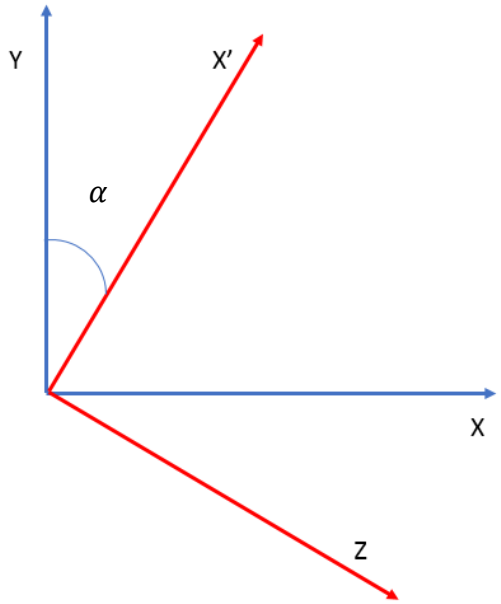


Fig.5.6: Representación ejes de coordenadas Ground Truth

Los ejes X e Y son los correspondientes a la latitud y longitud, y los ejes X' y Z los ejes del *Ground Truth*.

Por tanto, la equivalencia entre ambos ejes que forman un ángulo α es la siguiente:

$$X = X' \cdot \sin\alpha + Z \cdot \cos\alpha \tag{5.1}$$

$$Y = X' \cdot \cos\alpha - Z \cdot \sin\alpha \tag{5.2}$$

Para averiguar el ángulo α se han utilizado varios puntos del mapa, calculando su correspondiente X e Y y obteniendo del fichero *Ground Truth* su X' y Z, dando como resultado un ángulo de 31º.

Una vez conocida la equivalencia entre ambos ejes, se ha programado una lectura de fichero de texto para que se obtengan los valores X' y Z de la imagen correspondiente y se conviertan en los ejes X e Y, calculando así el punto donde está ubicado realmente el vehículo según la secuencia de imágenes.

Finalmente, el resultado obtenido es mostrado a continuación (*Fig. 5.7)*, donde el punto de referencia utilizado es el mismo que el del apartado anterior:

```
La ubicacion media es: Latitud=48.985643 Longitud=8.394096
La ubicacion original es: Latitud=48.985644 Longitud=8.394088
El error medio es: 0.284814 metros
```

Fig. 5.7: Ubicación obtenida

En esta imagen, el primer resultado que se muestra es la ubicación media calculada por el programa, en latitud y longitud. A continuación, se muestra la ubicación obtenida por el *Ground Truth* y por último el error medio entre todas las ubicaciones calculadas y la original.

La representación de la ubicación obtenida se muestra en la *Fig. 5.8*:

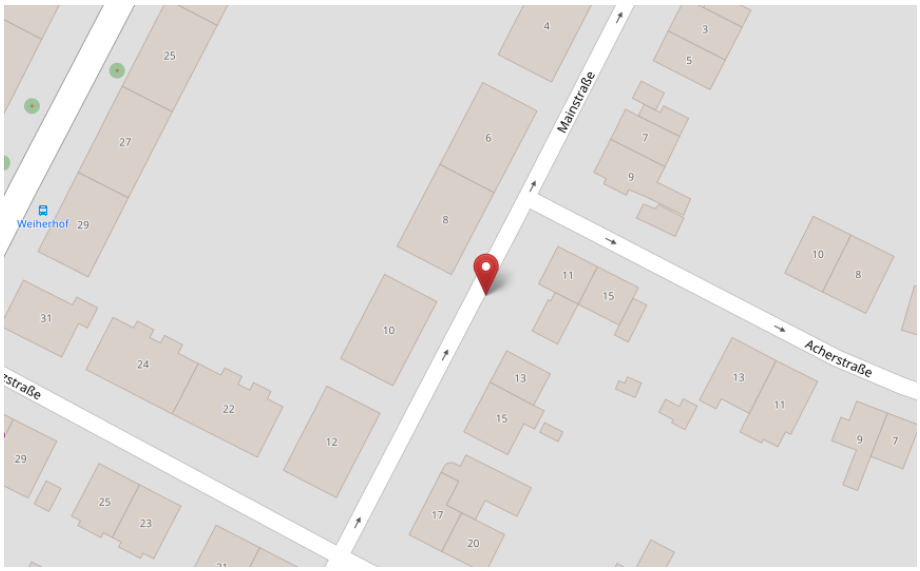


Fig. 5.8: Representación de la ubicación obtenida

Como se puede comprobar, ambas ubicaciones son muy parecidas, obteniendo un error de 0,284814 metros. Por tanto, se puede concluir que el código programado funciona correctamente, pudiendo modificar los valores de los umbrales para calibrar el cálculo de la ubicación.

En este primer caso se ha utilizado un umbral de error de 2 metros en la distancia y de 0.00003 grados en latitud y longitud. Esto significa que los puntos de corte deben pertenecer a los siguientes intervalos para ser aceptables, teniendo un error menor a 2 metros en distancia:

$$latitud = 48.985644° \pm 0.00003° \quad (5.3)$$

$$longitud = 8.394088° \pm 0.00003° \quad (5.4)$$

Si se modifican los umbrales a 5 metros y 0.00005 grados, el resultado obtenido es el siguiente (*Fig. 5.9)*:

```
La ubicacion media es: Latitud=48.985643 Longitud=8.394075
La ubicacion original es: Latitud=48.985644 Longitud=8.394088
El error medio es: 0.623545 metros
```

Fig. 5.9: Ubicación obtenida con modificación de umbrales

Como se puede observar, en este caso el resultado obtenido presenta un error superior al anterior, lo cual era lo esperado ya que las condiciones eran menos restrictivas.

Por tanto, una vez que se ha comprobado el correcto funcionamiento del programa, se realizaron pruebas a lo largo de todo el recorrido de la secuencia con el fin de verificar el funcionamiento en todos los puntos.

Sin embargo, se encontraron situaciones en las que el programa no funciona, numeradas a continuación:

1. Carreteras o tramos de carreteras en las que los edificios están pegados unos a otros, y por tanto no existen planos perpendiculares.
2. Curvas e intersecciones.

En el primer caso, dado que el método programado emplea los planos perpendiculares y paralelos de los edificios, únicamente se puede obtener información de la dirección de la carretera en la que está el vehículo, pero no su ubicación al no poder calcular ningún punto de corte.

Fig. 5.10: Ejemplo de carretera sin planos perpendiculares

Respecto al segundo caso, la librería Point Cloud utiliza los ejes del vehículo como los ejes de referencia a la hora de realizar la segmentación de planos. Por tanto, al estar situado en una intersección los ejes del vehículo están rotados respecto a los que se utilizan como condición en la segmentación, sin obtener resultado alguno.

Estos casos serán analizados en el siguiente experimento, con el fin de poder encontrar una solución.

5.3 Experimento 3: Mapeado de la secuencia número 7

El objetivo de este tercer experimento es el de mapear toda la secuencia con la ubicación del vehículo para poder comprobar si se obtienen buenos resultados.

Para ello, es necesario ampliar el código del apartado anterior para implementar un bucle, el cual calcule de forma iterativa la posición a lo largo de la secuencia en vez de en un único punto en concreto.

Debido a los casos en los que el código no funciona, primero se ha mapeado la secuencia sin tener en cuenta la excepción número 3 del apartado anterior.

Además, se han guardado los valores de los errores medios cometidos para representarlos posteriormente en una gráfica.

Por tanto, el resultado obtenido es el mostrado en la *Fig. 5.11*:

Fig.5.11: Mapeado parcial de la secuencia número 7

En la *Fig. 5.11* se observa el mapeado parcial de la secuencia número 7, utilizando el editor de mapas gratuito de OpenStreetMap.

Para ello, se ha analizado el resultado de cada calle por separado, calibrando los valores obtenidos en cada una de ellas con los umbrales de error para minimizar el error obtenido, ya que existen zonas en las que es necesario aumentar estos umbrales al estar los edificios más separados de lo que están en realidad, como se mencionó en el primer experimento.

Un claro ejemplo de este caso es la carretera etiquetada con el número 1 en amarillo, donde se observa que los puntos obtenidos forman una recta paralela a la carretera, pero desplazada.

Respecto al caso de las carreteras sin planos perpendiculares, como se mencionó en el apartado anterior no es posible determinar la ubicación del vehículo, únicamente su dirección.

Sin embargo, para que funcione el programa y sea capaz de ubicar el vehículo de forma correcta al detectar el primer plano perpendicular ha sido necesario programar un código diferente.

El primer paso es calcular la velocidad del vehículo en esa carretera, la cual se considerará constante en todo el tramo.

Para poder calcular esta velocidad, cada secuencia de imágenes contiene un archivo de texto en el cual están anotados los tiempos que transcurren entre cada imagen en la secuencia. Con estos datos de tiempo, se calcula la distancia recorrida entre dos ubicaciones obtenidas por el programa y se obtiene la velocidad mediante la siguiente ecuación:

$$v = \frac{\Delta d}{\Delta t} \tag{5.5}$$

A continuación, se guarda la posición obtenida en latitud y longitud del último punto en el que se detectan planos perpendiculares, además de su número de la imagen, y se continúa ejecutando el código, hasta que se vuelve a detectar otro plano perpendicular.

Cuando esto ocurre, se calcula la distancia entre ambos puntos mediante:

$$\Delta d = v \cdot \Delta t \tag{5.6}$$

Finalmente, a partir de la distancia calculada y de la dirección de la carretera se descompone en distancia en eje X y distancia en eje Y para obtener la latitud y longitud de este último punto.

Resuelto este problema, el siguiente objetivo es poder ubicar el vehículo en las curvas e intersecciones.

Como se mencionó en el apartado anterior, los ejes del vehículo son los ejes de referencia para el cálculo de la segmentación. Por tanto, cuando el vehículo está en paralelo a la carretera el código funciona correctamente. Sin embargo, al realizar una curva estos ejes rotan y no se detectan los planos ya que las condiciones impuestas son que se detecten planos del tipo $(0,1,0)$ y $(1,0,0)$.

Para solucionar este problema, es necesario modificar el código para que, al entrar en una curva, se obtengan imagen a imagen los distintos planos y se utilice la ecuación de estos planos como coeficientes que sustituyan a los anteriores.

Por ejemplo, como el umbral de ángulo es de 10º, se detectan todos los planos que tengan la misma orientación que los impuestos con un error de 10º. Cuando el vehículo empiece a girar, los planos tendrán coeficientes de la forma (0.1, 0.98 ,0) y (0.98, 0.1, 0), es decir, estarán girados respecto a los coeficientes originales.

Con estos valores, se sustituyen los coeficientes originales y se vuelve a calcular para la siguiente imagen de forma iterativa pudiendo modificar la orientación de los planos que se desean detectar.

Finalmente, el resultado obtenido al aplicar esta modificación al código en las curvas e intersecciones es el siguiente (*Fig. 5.12)*:

Fig. 5.12: Mapeado completo de la secuencia número 7

En esta imagen se muestra el mapeado completo de la secuencia número 7. Se puede observar la acumulación de puntos en las curvas ya que se ha calculado imagen a imagen la ubicación en estos casos.

Por tanto, se puede concluir que el código proporciona resultados correctos ya que el mapeado realizado sigue la forma de la secuencia con errores muy pequeños.

Para poder visualizar estos errores, a continuación se muestra un gráfico (*Fig. 5.13)* con el error medio de cada uno de los puntos:

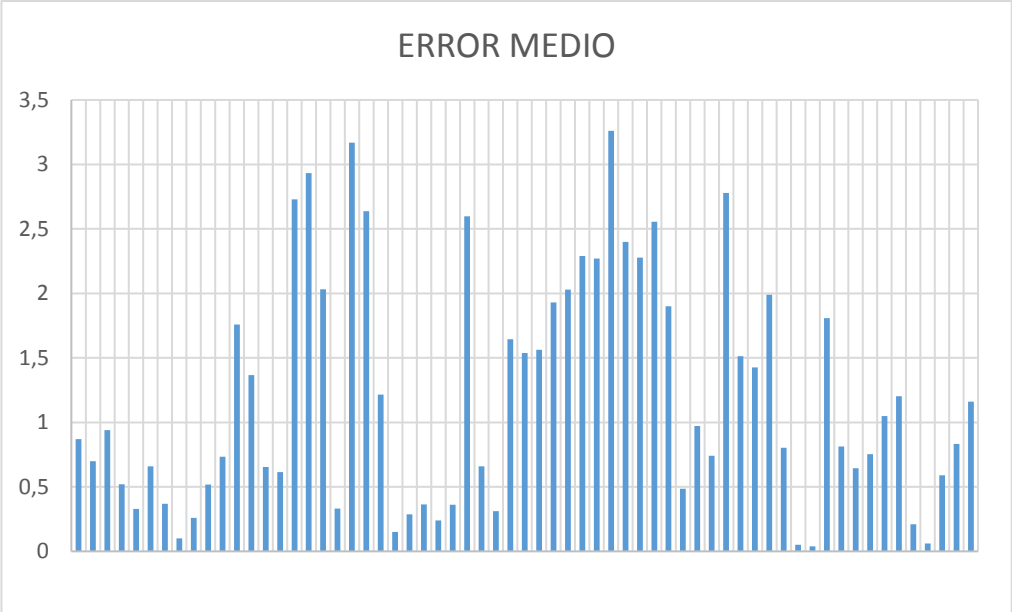


Fig. 5.13: Errores de la secuencia número 7

En este gráfico, en el eje Y se representa el valor del error medio en metros, con un valor máximo de 3,26, un valor mínimo de 0,05, y un valor medio de 1,206 metros.

Aunque en algunos puntos se obtengan valores elevados de errores, con el mapeado se observa que el resultado es correcto ya que la forma obtenida se corresponde con el recorrido del vehículo. Además, estos valores de errores incluyen el error proporcionado por OpenStreetMap.

Por tanto, el resultado del experimento es satisfactorio y a continuación se procede a ponerlo en práctica en la secuencia número 8.

5.4 Experimento 4: Mapeado de la secuencia número 8

En este cuarto y último experimento se pretende comprobar el funcionamiento total del proyecto en otra secuencia distinta, la secuencia número 8.

El motivo de haber elegido esta secuencia es porque, al igual que la secuencia número 7, contiene varias calles con edificios. Además, también proporciona datos de *Ground Truth* para poder comprobar el resultado obtenido.

El recorrido que sigue el vehículo es el mostrado en la *Fig. 5.14*:

Fig. 5.14: Recorrido secuencia número 8

Para poder aplicar el mismo método utilizado en el apartado anterior es necesario volver a calcular la equivalencia entre el *Ground Truth* y los ejes X e Y del mapa.

Para poder aplicar el mismo método utilizado en el apartado anterior es necesario volver a calcular la equivalencia entre el *Ground Truth* y los ejes X e Y del mapa.

En la *Fig. 5.15* se muestra la representación de ambos sistemas de coordenadas:

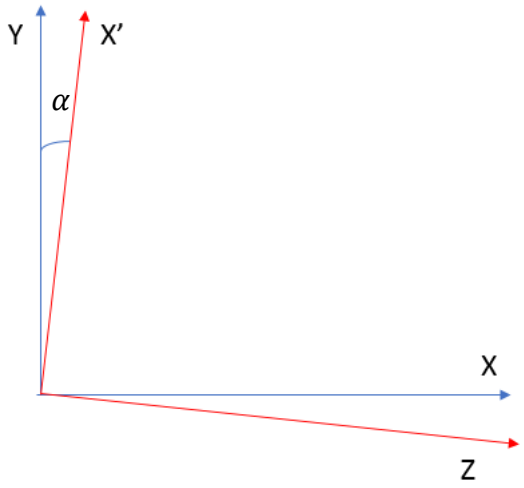


Fig. 5:15: Representación ejes de coordenadas secuencia 8

De esta forma, la equivalencia es la siguiente:

$$X = X' \cdot \sin\alpha + Z \cdot \cos\alpha \tag{5.7}$$

$$Y = X' \cdot \cos\alpha - Z \cdot \sin\alpha \tag{5.8}$$

Al igual que en la secuencia anterior, para averiguar el ángulo α se han utilizado diversos puntos del mapa, calculando su correspondiente X e Y y obteniendo del fichero *Ground Truth* su X' y Z, dando como resultado un ángulo de 6º.

Por tanto, conociendo esta equivalencia se puede comenzar a mapear para ver el resultado obtenido (*Fig. 5.16)*:

Fig. 5.16: Mapeado completo de la secuencia número 8

Como se puede observar en la imagen, el mapeado no se ha podido realizar en toda la secuencia. Esto es debido a dos motivos principales.

El primero tiene que ver con el problema obtenido en la anterior secuencia, donde no se podía obtener la ubicación del vehículo en calles con edificios juntos. En esta secuencia hay varias calles que presentan esta forma (números 1, 2 y 3), con hileras de edificios juntos y un espacio entre hileras donde sí se obtienen resultados.

El segundo es debido a un problema que no se presentó en la anterior secuencia, los obstáculos (calles números 3, 4 y 5).

En estas carreteras hay obstáculos, en este caso árboles, que impiden al láser detectar los edificios cercanos, por lo que no se puede determinar la ubicación del vehículo en estas condiciones.

Sin embargo, en las zonas donde sí se han obtenido resultados se puede comprobar que el funcionamiento es correcto, presentando los mismos errores que en la anterior secuencia provenientes de los datos de OpenStreetMap.

Al igual que en el apartado anterior, se han guardado los errores de cada punto, mostrados en la siguiente gráfica (*Fig. 5.17)*:

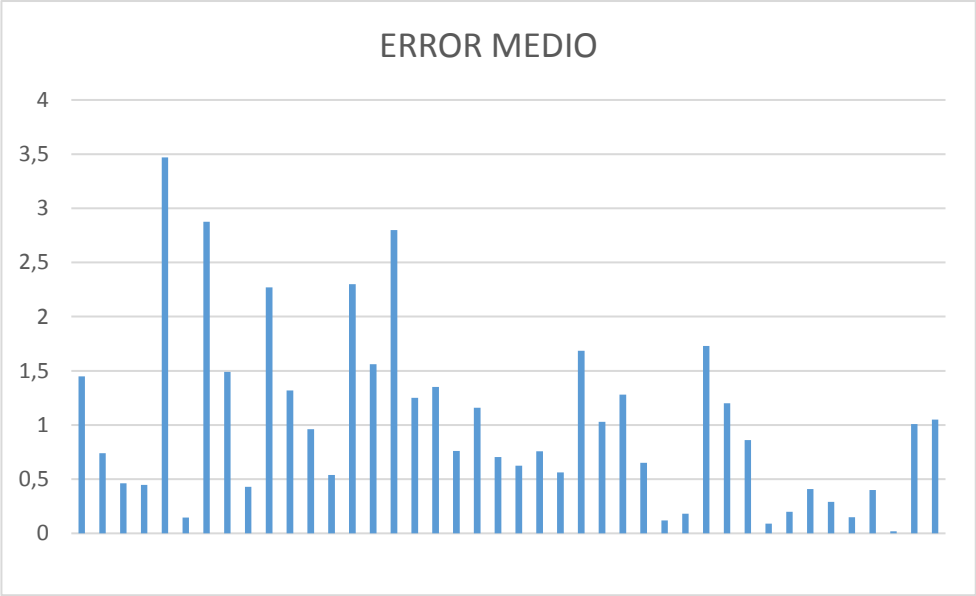


Fig.5.17: Errores de la secuencia número 8

En el eje Y se representa el error obtenido en metros, con un error máximo de 3,46, mínimo de 0,02 y error medio de 1,018 metros.

Por tanto, con estos resultados obtenidos se puede concluir que el funcionamiento del proyecto es correcto para las distintas secuencias, con las limitaciones mencionadas en ambos experimentos.

6. CONCLUSIONES

En este proyecto se ha implementado un código en lenguaje C++ que permite localizar un vehículo autónomo circulando en entornos urbanos.

Para ello se han utilizado varias herramientas como la librería de C++ Libosmium (para los datos de OpenStreetMap) y Point Cloud Library, además de los datos del proyecto KITTI Vision Benchmark Suite.

Durante el desarrollo del proyecto se han realizado distintas pruebas, explicadas en los apartados 4 y 5 de la memoria, las cuales comenzaron siendo pruebas muy simples para aprender a manejar tanto el lenguaje como las herramientas, para ir aumentando la dificultad del programa conforme los resultados eran los deseados.

Primero se probaron por separado los datos de OpenStreetMap utilizando la librería Libosmium y los datos de KITTI mediante Point Cloud Library, para juntarlos una vez se supieran utilizar.

Esto causó numerosos problemas ya que ambas librerías se compilaban de forma distinta, y al juntarlas no se podía ejecutar el programa. Para solucionar este problema se recurrió a la documentación de Libosmium donde se explicaba en un archivo de texto cómo incluir esta librería en otras distintas.

Una vez solucionado el problema se comenzaron a obtener resultados válidos para el proyecto, primero probando a obtener la ubicación del vehículo en un punto y más tarde en la secuencia entera.

Sin embargo, a la hora de ubicar el vehículo en las curvas el programa no proporcionaba ningún resultado, al igual que en carreteras con hileras de edificios juntos.

Para este último caso se llegó a la conclusión de que no se puede obtener la ubicación mediante este método, únicamente la dirección de desplazamiento del vehículo hasta que hubiera espacios entre edificios o se llegase a una intersección, lo cual a efectos prácticos no es un problema muy grave ya que el vehículo seguiría circulando a lo largo de la misma carretera sabiendo que no hay intersecciones cerca.

Por el otro lado, se consiguió solucionar el problema de las curvas actualizando los ejes de búsqueda de la segmentación de planos con los ejes del propio vehículo.

Otro problema que se encontró en el cual no se obtenía resultado de ubicación fue en lugares donde hubiera obstáculos que impidieran la detección de los edificios por el láser del vehículo. Esto se puede solucionar y a la vez no, ya que existen obstáculos dinámicos como un camión, que únicamente impiden la detección durante un periodo de tiempo relativamente corto y obstáculos estáticos como árboles que están durante mucho más tiempo.

Teniendo en cuenta estos casos de conflicto, el resultado final del proyecto se puede considerar satisfactorio, obteniendo errores medios de ubicación alrededor de 1 metro a lo largo de una secuencia completa.

Estos errores podrían disminuirse con datos más precisos del mapa de la ciudad, ya que como se ha comentado en otros apartados OpenStreetMap en algunas ocasiones presenta un error en la localización de los edificios.

Respecto a las dificultades encontradas, comencé a desarrollar este proyecto en el mes de noviembre sin tener conocimiento previo del lenguaje C++. Esto me obligó a avanzar muy lentamente, ya que tuve que consultar mucha documentación tanto del lenguaje como de las librerías utilizadas.

Sin embargo, en el segundo cuatrimestre elegí una asignatura de programación en C++ la cual me permitía avanzar de una forma mucho más rápida.

Otra de las dificultades que me han surgido a lo largo de este tiempo es la falta de experiencia en utilizar el sistema operativo Linux. En instalar las librerías y aprender a compilar los programas invertí una gran cantidad de tiempo al ser un sistema operativo totalmente nuevo para mí.

Además, como recursos he utilizado mi ordenador personal, y al no saber cómo instalar Linux junto con Windows decidí hacer uso de una máquina virtual, la cual consumía una gran cantidad de memoria del ordenador. Por tanto, en numerosas ocasiones se quedaba bloqueado el sistema perdiendo los progresos realizados, y las compilaciones de los programas llegaron a tardar más de 3 minutos, impidiendo un avance al ritmo que tenía pensado.

Gracias a la documentación disponible de Libosmium, Point Cloud Library y KITTI, y a la ayuda de mi tutor, pude resolver todas las dificultades que me fueron surgiendo en estos meses y completar el proyecto.

Por tanto, la elaboración de este proyecto me ha permitido introducirme en el área de los vehículos autónomos, pudiendo conocer sus componentes tanto hardware como software de una forma más amplia.

Además, me ha dado la oportunidad de aprender un nuevo lenguaje de programación y de utilizar herramientas que desconocía.

Por todo esto, pienso que la elaboración de este proyecto me ha proporcionado muchos conocimientos importantes que no tenía, al igual que me ha permitido cambiar mi forma de trabajo teniendo que desarrollar algo que era nuevo para mí, buscando y obteniendo información en la documentación de los recursos empleados.

7. TRABAJOS FUTUROS

En este apartado se proponen ideas que podrían implementarse en el futuro para aumentar las funcionalidades y utilidades de este proyecto:

- Utilizar distintos métodos de segmentación de la nube de puntos, comparando los resultados obtenidos. Como cada método utiliza un algoritmo diferente de segmentación, podría darse el caso de que se obtuvieran errores menores que con el utilizado en este proyecto (*RANSAC).*
- Utilizar datos de distintos mapas libres para comparar los resultados de ubicación. Al igual que en el punto anterior, podrían haber otros mapas que proporcionasen menores errores en la ubicación de los edificios, disminuyendo el error final en la ubicación.
- Implementación del programa en el vehículo autónomo, para lo cual sería necesario utilizar la herramienta ROS (*Robot Operating System*). Con esto se podría comprobar el funcionamiento del proyecto en tiempo real, pudiendo recoger datos para mejorar y corregir el código.
- Implementación del proyecto en las demás secuencias. Las secuencias elegidas en esta memoria contenían numerosas calles con edificios donde se pueden obtener resultados correctos. Sin embargo, otras secuencias no tenían tantos edificios. Además, la mitad de ellas no disponían de Ground Truth para comprobar el resultado. Por ello, implementar el proyecto en estas secuencias podría ser interesante para comprobar el funcionamiento en todas ellas.

8. BIBLIOGRAFÍA

[1] V. Milanés *et al*, "Sistema de Posicionamiento para Vehículos Autónomos". *Revista Iberoamericana de Automática e Informática industrial*, *5*(4), 36-41, Octubre de 2008.

[2] DGT," Tráfico establece el marco para la realización de pruebas con vehículos de conducción automatizada en vías abiertas a la circulación", 16-11-2015. [En línea]. Disponible en: http://www.dgt.es/es/prensa/notas-de-prensa/2015/20151116-trafico-establece-marco-realizacion-pruebas-vehiculos-conduccion-automatizada-vias-abiertas-circulacion.shtml . Acceso: 10-06-2018

[3] A.Vigil Hochleitner, "¿Qué leyes habría que cambiar antes de que llegue el coche autónomo?", 28-09-2017. [En línea]. Disponible en: https://cincodias.elpais.com/cincodias/2017/08/07/legal/1502093470_669276.html. Acceso: 10-06-2018

[4] P. G. Bejerano, "La desconocida historia de los coches autónomos" 12-09-2013. [En línea]. Disponible en: https://blogthinkbig.com/historia-de-los-coches-autonomos. . Acceso: 10-06-2018

[5] National Geographic, "Pasado, presente y futuro de la conducción autónoma" 31-10-2017. [En línea]. Disponible en: http://www.nationalgeographic.com.es/promociones/pasado-presente-futuro-conduccion-autonoma_12014. Acceso: 10-06-2018

[6] C. Sánchez, "Ernst Dickmanns, el desconocido padre alemán de los coches inteligentes" 30-04-2015. [En línea]. Disponible en: https://www.eldiario.es/hojaderouter/tecnologia/Ernst_Dickmanns-vehiculo-autonomo-inteligente_0_382511814.html. Acceso: 10-06-2018

[7] K. Hyatt, C. Paukert, "Los vehículos autónomos: entendiendo los distintos niveles de automatización" 01-04-2018. [En línea]. Disponible en: https://www.cnet.com/es/noticias/vehiculos-autonomos-niveles-de-automatizacion/. Acceso: 10-06-2018

[8] The KITTI Vision Benchmark Suite. [En línea]. Disponible en: http://www.cvlibs.net/datasets/kitti/. Acceso: 10-06-2018

[9] L. del Valle Hernández, "Coches autónomos, el estado del arte con Alex Barredo" [En línea]. Disponible en: https://programarfacil.com/podcast/coche-autonomo-estado-del-arte/. Acceso: 10-06-2018

[10] P. Ibáñez "Qué es un LIDAR, y cómo funciona el sensor más caro de los coches autónomos" 28-09-2017. [En línea]. Disponible en: https://www.motorpasion.com/tecnologia/que-es-un-lidar-y-como-funciona-el-sistema-de-medicion-y-deteccion-de-objetos-mediante-laser. Acceso: 10-06-2018

[11] S. Holzer, N. Blodow, "PCL::Segmentation", Noviembre de 2011.

[12] GitHub. [En línea]. Disponible en: https://github.com/osmcode/libosmium. Acceso: 10-06-2018

[13] [En línea]. Disponible en: https://extract.bbbike.org/. Acceso: 10-06-2018

[14] [En línea]. Disponible en: http://www.cvlibs.net/datasets/kitti/eval_odometry.php. Acceso: 10-06-2018